International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2013): 6.14 | Impact Factor (2015): 6.391

Design of a More Efficient and Effective Flip Flop use of K-Map Based Boolean Function

M. Valli[1], Dr. R. Periyasamy[2]

[1]Assistant Professor, St. Joseph College of Arts & Science (Autonomous), Cuddalore. Tamilnadu, India

[2]Associate Professor, Nehru Memorial College (Autonomous), Trichy, Tamilnadu, India

Abstract: ***In this paper, we present a novel method to efficiently process SR flip flop spatial queries with conjunctive Boolean constraints on textual content. Our method combines an R-tree with an inverted index by the inclusion of spatial references in posting lists. The result is a disk resident, dual-index data structure that is used to proactively prune the search space. R-tree nodes are visited in best-first order. A node entry is placed in the priority queue if there exists at least one object that satisfies the Boolean condition in the sub tree pointed by the entry; otherwise, the Sub tree is not further explored. We show via extensive experimentation with real spatial databases that our method has increased performance over alternate techniques while scaling to large number of objects.***

Keywords: Boolean Circuit, K Map, Flip flop, Gate, Algebra.

1. Introduction

Boolean algebra forms a cornerstone of computer science and digital system design. Many problems in digital logic design and testing, artificial intelligence, and combinatory can be expressed as a sequence of operations on Boolean functions. Such applications would benefit from efficient algorithms for representing and manipulating Boolean functions symbolically. Unfortunately, many of the tasks one would like to perform with Boolean functions, such as testing whether there exists any assignment of input variables such that a given Boolean expression evaluates to 1 , or two Boolean expressions denote the same function (equivalence) require solutions to NP-Complete or Complete problems . Consequently, all known approaches to performing these operations require, in the worst case, an amount of computer time that grows exponentially with the size of the problem. This makes it difficult to compare the relative efficiencies of different approaches to representing and manipulating Boolean functions. In the worst case, all known approaches perform as poorly as the naive approach of representing functions by their truth tables and defining all of the desired operations in terms of their effect on truth table entries. In practice, by utilizing more clever representations and manipulation algorithms, we can often avoid these exponential computations. Flip-Flops are digital circuits with two stable, self-maintaining states that are used as storage/ memory elements such as Random Access Memory, Caches Memory and Read Only Memory. They are also very useful in the following electronic digital devices design; Sequence Detector, Data Synchronizer, Frequency Divider, Registers, Counters and Registers in Central Processing Unit for data transfer. They are derived from Sequential Logic Circuits which are the main electronics circuits that make the development of computers possible. The ability of computer systems to operate without the continuous human intervention is solely achieved through sequential logic circuits, the building blocks of Flip Flops. With the growing popularity of portable devices, power reduction has become a popular design goal for advanced design application, whether mobile or not. Reducing power consumption in chips enables better, cheaper products to be designed and power-related chip failures to be minimized. As a result, how to minimize power consumption has become an important design goal that every chip designer must take care. Several lower power design techniques have played an important role in the design flow. Clock gating methodology is used for the register bank to replace the multiplexers and it can avoid the operation of reloading the same data value. The clock gating technique could reduce the dynamic power consumption efficiently. The multi-V_{th} concept is aimed at using multi-V_{th} cell with satisfying performance to reduce leakage consumption, and replace lower V_{th} (LVT) cells by high V_{th} (HVT) ones, if there is room for slack. Multiple Supply Multiple Voltage (MSMV) supplies of different voltages are used for core logic, base on satisfy performance or functional requirement to adjust operating voltage for each domain, even shut off this domain.

Given a design that the locations of the cells have been determined; the power consumed by clocking can be reduced further by replacing several flip-flops with multi-bit flip-flops. During clock tree synthesis, less number of flip-flops means less number of clock sinks. Thus, the resulting clock network would have smaller power consumption and uses less routing resource. Multi-bit flip-flop is an effective power-saving implementation methodology by merging single-bit flip-flops in the design. Using multi-bit flip-flops can reduce clock dynamic power and the total flip-flop area effectively

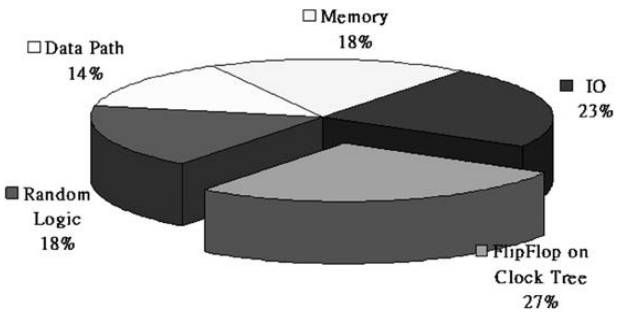


Figure 1: Example of ASIC chip power distribution

Figure 1 is an ASIC chip power distribution. We can see that the flip-flops on clock tree accounted for a large proportion of power consumption. Although the power distribution will vary with different ASIC design, reducing power consumption of the flip-flop on clock tree can eliminate total power consumption efficiently.

1.1 Motivation for the study

It has been observed that computer performance is primarily affected by the processor and memory. If either one reaches its limits (which may initially be the memory), the performance of the whole system degrades. As semiconductor technology advances, the performance gap between processor - the Central Processing Unit and main memory - the Random Access Memory has become one of the major issues in computer design. In the past 35 years, an exponential rate of improvement has been witnessed in semiconductor technology. This imbalance has become one major bottleneck in further improving the computer performance. One reason memory system performance has consistently lagged processor performance is that memory systems typically consist of one or more chips that are designed and manufactured separately from the processor, and the performance of the interconnected multi-chip memory system is difficult to scale to achieve higher data rate and lower access latency. Memory system data rates are increasing with each new generation of memory devices at the rate of 100% every three years, and memory row cycle times. The collective trends are increasing the ratio of row cycle times to the duration of data bursts on the data bus. This is why it is imperative to critically evaluate the existing conventional SR-FF and the need to bridge the speed gap between memory and processor by enhancing the memory speed through logical modification frameworks of the conventional SR-FF which utilizes states. With the development of information technology, data volumes processed by many applications will routinely cross the scale threshold, which would in turn increase the computational requirements. Efficient parallel clustering algorithms and implementation techniques are the key to meeting the scalability and performance requirements entailed in such scientific data analyses. So far, several researchers have proposed some parallel clustering algorithms. They assume that all objects can reside in main memory at the same time; b) their parallel systems have provided restricted programming models and used the restrictions to parallelize the computation automatically. Both assumptions are prohibitive for very large datasets with millions of objects. Therefore, dataset oriented parallel clustering algorithms should be developed. Map Reduce is a programming model and an associated implementation for processing and generating large datasets that is amenable to a broad variety of real-world tasks. Users specify the computation in terms of a map and a reduce function, and the underlying runtime system automatically parallelizes the computation across large-scale clusters of machines, handles machine failures, and schedules inter-machine communication to make efficient use of the network and disks. It Reduce runtimes with fault tolerance and dynamic flexibility support .In this paper, we adapt *k*-means algorithm in map Reduce framework which is implemented by the clustering method applicable to large scale data. We conduct comprehensive experiments to evaluate the proposed algorithm. The results demonstrate that our algorithm can effectively deal with large scale datasets. The rest of the paper is organized as follows. We presented our parallel *k*-means algorithm based on Map Reduce framework. In this paper, we propose an efficient method for solving systems of Boolean equations. Rather than confining ourselves to the Boolean domain, our idea is to convert the problem so that we operate in the integer domain. The integer domain is a richer domain to work with, as algorithms there are well-developed.

2. Related Works

Efficiently Evaluating Complex Boolean Expressions , Marcus Fontoura, 2010,The problem of efficiently evaluating a large collection of complex Boolean expressions – beyond simple conjunctions and Disjunctive/Conjunctive Normal Forms (DNF/CNF)– occurs in many emerging online advertising applications such as advertising exchanges and automatic targeting. The simple solution of normalizing complex Boolean expressions to DNF or CNF form, and then using existing methods for evaluating such expressions is not always effective because of the exponential blow-up in the size of expressions due to normalization. We thus propose a novel method for evaluating complex expressions, which leverages existing techniques for evaluating leaf-level conjunctions, and then uses a bottom-up evaluation technique to only process the relevant parts of the complex expressions that contain the matching conjunctions. We develop two such bottom-up evaluation techniques, one based on Dewey IDs and another based on mapping Boolean expressions to one-dimensional intervals. Our experimental evaluation based on data obtained from an online advertising exchange shows that the proposed techniques are efficient and scalable, both with respect to space usage as well as evaluation time.

A Shannon Based Low Power Adder Cell for Neural Network Training K.Nehru, 2010,The proposed full adders for low power and high performance neural network training circuits has been implemented using Shannon decomposition based technique for sum and carry operation. The hardware includes multiplier circuit for product term and an adder circuit to perform summation. The proposed full adder is designed using tanner EDA tools and the resulting parameters such as 25.6% improvement in power dissipation and 20% improvement in transistor count from the simulated output when compared with MC IT based adder cell.

Debajit Sensarma, On An Optimization Technique Using Binary Decision Diagram, 2012, Two-level logic minimization is a central problem in logic synthesis, and has applications in reliability analysis and automated reasoning. This paper represents a method of minimizing Boolean sum of products function with binary decision diagram and with disjoint sum of product minimization. Due to the symbolic representation of cubes for large problem instances, the method is orders of magnitude faster than previous enumerative techniques. But the quality of the approach largely depends on the variable ordering of the underlying BDD. The application of Binary Decision Diagrams (BDDs)

as an efficient approach for the minimization of Disjoint Sums-of-Products (DSOPs). DSOPs are a starting point for several applications. The use of BDDs has the advantage of an implicit representation of terms. Due to this scheme the algorithm is faster than techniques working on explicit representations and the application to large circuits that could not be handled so far becomes possible. Theoretical studies on the influence of the BDDs to the search space are carried out. In experiments the proposed technique is compared to others. The results with respect to the size of the resulting DSOP are as good or better as those of the other techniques.

Randal E. Bryant, Graph-Based Algorithms For Boolean Function Manipulation, 2008, in this paper we present a new data structure for representing Boolean functions and an associated set of manipulation algorithms. Functions are represented by directed, acyclic graphs in a manner similar to the representations introduced but with further restrictions on the ordering of decision variables in the graph. Although a function requires, in the worst case, a graph of size exponential in the number of arguments, many of the functions encountered in typical applications have a more reasonable representation. Our algorithms have time complexity proportional to the sizes of the graphs being operated on, and hence are quite efficient as long as the graphs do not grow too large. We present experimental results from applying these algorithms to problems in logic design verification that demonstrate the practicality of our approach.

D. Markovic, A General Method In Synthesis Of Pass-Transistor Circuits, 2000, A general method in synthesis and signal arrangement in different pass-transistor network topologies is analyzed. Several pass-transistor logic families have been introduced recently, but no systematic synthesis method is available that takes into account the impact of signal arrangement on circuit performance. In this paper we develop a Karnaugh map based method that can be used to efficiently synthesize pass transistor logic circuits, which have balanced loads on true and complementary input signals. The method is applied to the generation of basic two-input and three-input logic gates in CPL, DPL and DVL. The method is general and can be extended to synthesize any pass-transistor network.

Weizhong Zhao, **BlastReduce: High Performance Short Read Mapping with MapReduce,2007,**Next-generation DNA sequencing machines generate sequence data at an unprecedented rate, but traditional single-processor sequence alignment algorithms are struggling to keep pace with them. Blast Reduce is a new parallel read mapping algorithm optimized for aligning sequence data from those machines to reference genomes, for use in a variety of biological analyses, including SNP discovery, genotyping, and personal genomics. It is modeled after the widely used BLAST sequence alignment algorithm, but uses the open-source Hadoop implementation of Map Reduce to parallelize execution to multiple compute nodes. To evaluate its performance, Blast Reduce was used to map next generation sequence data to a reference bacterial genome in a variety of configurations.

Alan Mishchenko, Fast Heuristic Minimization of Exclusive-Sums-of-Products, Exclusive-Sums-Of-Products (ESOPs) play an important role in logic synthesis and design-for-test. This paper presents an improved version of the heuristic ESOP minimization procedure proposed . The improvements concern three aspects of the procedure :(1) computation of the starting ESOP cover; increase of the search space for solutions by applying a larger set of cube transformations; development of specialized data structures for robust manipulation of ESOP covers. Comparison of the new heuristic ESOP minimize EXORCISM-4 with other minimizes (EXMIN2 , MINT , EXORCISM and EXORCISM show that, in most cases, EXORCISM-4 produces results of comparable or better quality on average ten times faster.

3. Problem Statement

For Boolean logic functions of variables more than six, it is difficult to handle the minimization process using K-map technique. To get optimum solution one must be sure that the best selection has been made. The tabulation method overcomes this difficulty. It is a specific step-by-step procedure that is guaranteed to produce a simplified standard-form expression for a Boolean logic function. It can be applied to problems with many variables and has the advantage of being suitable for machine computation. However, it is quite tedious and is prone to mistakes because of its routine, and it is a monotonous process. It is cumbersome to manipulate Boolean expressions by hand, so a tool to verify the results is helpful. We developed a "minimization function checker" which verifies the correctness of the minimization results of Boolean logic functions.

4. Proposed Work

4.1 Flip flop based Boolean function

In this, we present a novel method to efficiently process SR flip flop spatial queries with conjunctive Boolean constraints on textual content. Our method combines an R-tree with an inverted index by the inclusion of spatial references in posting lists. The result is a disk resident, dual-index data structure that is used to proactively prune the search space. The nodes are visited in best-first order. A node entry is placed in the priority queue if there exists at least one object that satisfies the Boolean condition in the sub tree pointed by the entry; a novel method for evaluating complex expressions, which leverages existing techniques for evaluating leaf-level conjunctions, and then uses a bottom-up evaluation technique to only process the relevant parts of the complex expressions that contain the matching conjunctions. We develop two such bottom-up evaluation techniques, one based on Dewey IDs and another based on mapping Boolean expressions to one-dimensional intervals. Our experimental evaluation based on data obtained from an online advertising exchange shows that the proposed techniques are efficient and scalable, both with respect to space usage as well as evaluation time. In the integer domain better and more efficient methodologies for solving equations are available.

The conversion leads us to a system of polynomial equations obeying certain characteristics. A method is proposed for solving these equations. The most computationally demanding step is the repeated multiplication of polynomials. We develop a method for this problem that is significantly faster than the standard approach. We also introduce another variant of the method, the so-called hybrid approach that leads to reduced memory requirements. Such applications would benefit from efficient algorithms for representing and manipulating Boolean functions symbolically. Unfortunately, many of the tasks one would like to perform with Boolean functions, such as testing whether there exists any assignment of input variables such that a given Boolean expression evaluates to 1.

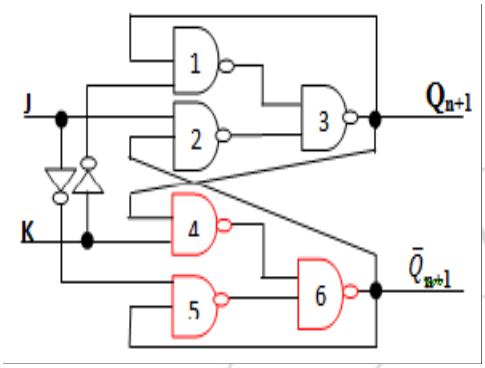


Figure 1: Logic Circuit of Conventional SR-Flip flop

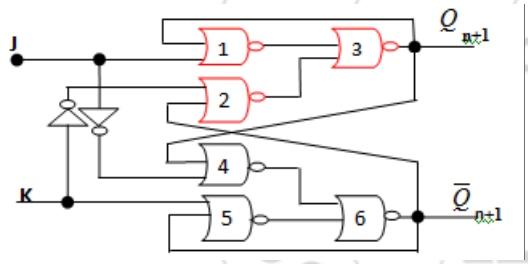


Figure 2: Logic Circuit of Conventional SR-Flip Flop

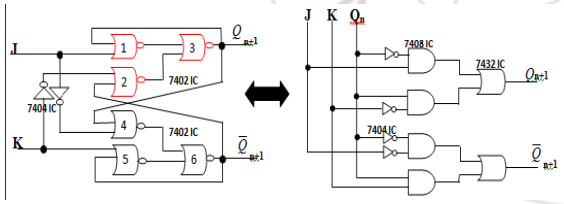


Figure 3: NOR gates SR-FF based on K-map

4.2 Module Description

4.2.1 Single bit Flip Flop

A single-bit flip-flop has two latches (Master latch and slave latch). The latches need "Clk" and "Clk‟ " signal to perform operations, such as Figure2 shows. In order to have better delay from Clk-> Q, we will regenerate "Clk" from "Clk‟ ". Hence we will have two inverters in the clock path.

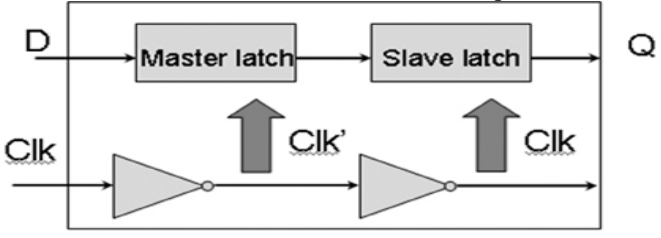


Figure 2: Single bit flipflop

4.2.2 Merging of Flip Flop

Figure 3 shows an example of merging two 1-bit flip-flops into one 2-bit flip-flop. Each 1-bit flip-flop contains two inverters, master-latch and slave-latch. Due to the manufacturing rules, inverters in flip-flops tend to be oversized. As the process technology advances into smaller geometry nodes like 65nm and beyond, the minimum size of clock drivers can drive more than one flip-flop. Merging single-bit flip-flops into one multi-bit flip-flop can avoid duplicate inverters, and lower the total clock dynamic power consumption. The total area contributing to flip-flops can be reduced as well.

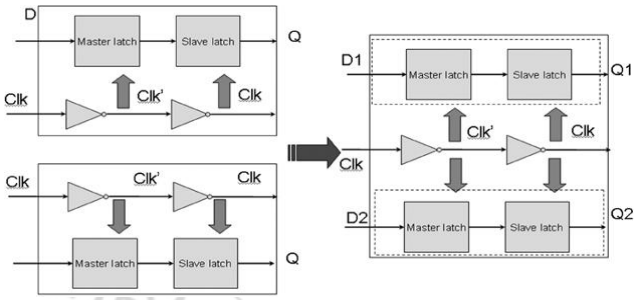


Figure 3: An example of merging two 1-bit flip-flops into one 2-bit flip-flop.

4.2.3 Multi bit Flip Flop

Figure 4.a shows an example of dual-bit flip-flop cell. It has two data input pins, two data output pins, one clock pin and reset pin. Use dual-bit flip-flop can get the benefits of lower power consumption then single-bit, and almost no other additional costs to pay. Figure 4.b shows the true table of dual-bit flip-flop cell. We could find that when CK is positive edge, the value of Q1 will pass to D1, and the value of Q2 will pass to D2. Or Q1 and Q2 will keep original value

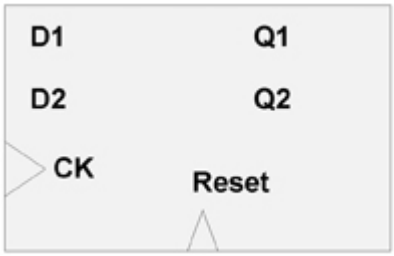


Figure 4 (a): A dual-bit flip-flop cell.

4.2 4 Transformation of Placement Space

We have shown that the shape of a feasible placement region associated with one pin *pi* connecting to a flip-flop *fi* would be diamond in previous analysis. Since there may exist several pins connecting to ***fi,***the legal placement region of ***f i*** are the overlapping area of several regions. As shown in Fig. 6(a), there are two pins *p*1 and *p*2 connecting to a flip-flop *f*1, and the feasible placement regions for the two pins are enclosed by dotted lines, which are denoted by *Rp (p*1) and *Rp(p2)*, respectively. Thus, the legal placement region *R (f1)* for *f*1 is the overlapping part of these regions. In Fig. 6(b), *R (f1)* and *R (f2)* represent the legal placement regions of *f*1 and *f*2. Because *R(f*1*)* and *R(f*2*)* overlap, we can replace *f*1 and *f*2 by a new flip-flop *f*3 without violating the timing constraint, as shown in Fig. 6(c).

However, it is not easy to identify and record feasible placement regions if their shapes are diamond. Moreover, four coordinates are required to record an overlapping region [see Fig. 7(a)]. Thus, if we can rotate each segment 45°, the shapes of all regions would become rectangular, which makes identification of overlapping regions become very simple. For example, the legal placement region, enclosed by dotted lines in Fig. 7(a), can be identified more easily if we change its original coordinate system [see Fig. 7(b)]. In such condition, we only need two coordinates, which are the left-bottom corner and right-top corner of a rectangle, as shown in Fig. 7(b), to record the overlapped area instead of using four coordinates. The equations used to transform coordinate system are shown in (1) and (2). Suppose the location of a point in the original coordinate system is denoted by (x, y). After coordinate transformation, the new coordinate is denoted by (x'', y').

$$x' = \frac{x+y}{\sqrt{2}} => x'' = \sqrt{2} \cdot x' = x + y \quad (1)$$

$$y' = \frac{-x+y}{\sqrt{2}} => y'' = \sqrt{2} \cdot y' = -x + y. \quad (2)$$

Then, we can find which flip-flops are merge able according to whether their feasible regions overlap or not. Since the feasible placement region of each flip-flop can be easily identified after the coordinate transformation, we simply use (3) and (4) to determine whether two flip-flops overlap or not.

$$\text{DIS\_}X(f_1, f_2) < \frac{1}{2}(W(f_1) + W(f_2)) \quad (3)$$

$$\text{DIS\_}Y(f_1, f_2) < \frac{1}{2}(H(f_1) + H(f_2)) \quad (4)$$

where $W(f1)$ and $H(f1)$ [$W(f2)$ and $H(f2)$] denote the width and height of $R(f1)$ [$R(f2)$], respectively, in Fig. 8, and the function DIS_X($f1$, $f2$) and (DIS_Y($f1$, $f2$)) calculates the distance between centers of $R(f1)$ and $R(f2)$ in x-direction (y-direction).

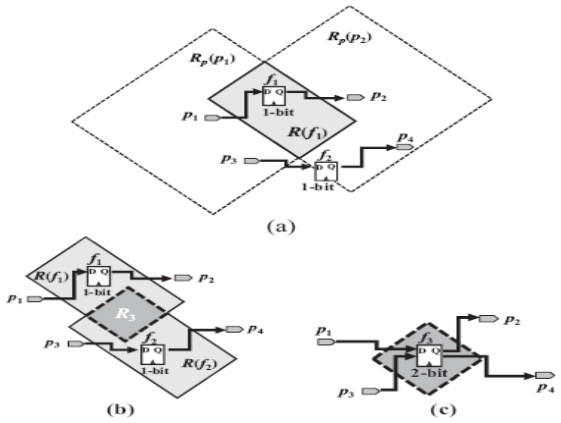


Figure 5: (a) Feasible regions $(p1)$ and $Rp(p2)$ for pins $p1$ and $p2$ which are enclosed by dotted lines, and the legal region $R(f1)$ for $f1$ which is enclosed by solid lines. (b) Legal placement regions $R(f1)$ and $R(f2)$ for $f1$ and $f2$, and the feasible area $R3$ which is the overlap region of $R(f1)$ and $R(f2)$. (c) New flip-flop $f3$ that can be used to replace $f1$ and $f2$ without violating timing constraints for all pins $p1$, $p2$, $p3$, and $p4$.

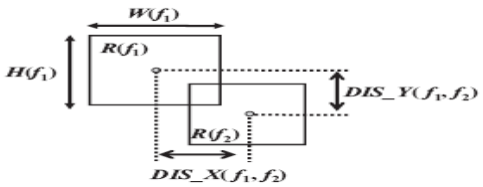


Figure 6: Overlapping relation between available placement regions of f 1 and f 2.

4.2.5 Build a Combination Table

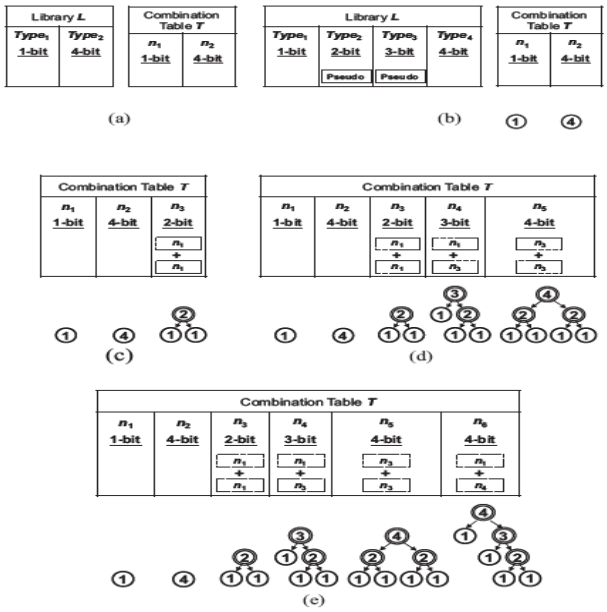
If we want to replace several flip-flops by a new flip-flop fi" (note that the bit width of f I should equal to the summation of bit widths of these flip-flops), we have to make sure that the New flip-flop fi" is provided by the library L when the feasible regions of these flip-flops overlap. In this paper, we will build a combination table, which records all possible combinations of flip-flops to get feasible flip-flops before replacements. Thus, we can gradually replace flip-flops according to the order of the combinations of flip-flops in this table. Since only one combination of flip-flops needs to be considered in each time, the search time can be reduced greatly.

We use a binary tree to represent one combination for simplicity. Each node in the tree denotes one type of a flip-flop in L. The types of flip-flops denoted by leaves will constitute the type of the flip-flop in the root. For each node, the bit width of the corresponding flip-flop equals to the bit width summation of flip-flops denoted by its left and right child [Fig. 9(e)]. Let ni denote one combination in T, and b(ni) denote its bit width. In the beginning, we initialize a combination ni for each kind of flip-flops in L (see Line 1). Then, in order to represent all combinations by using a binary tree, we may add pseudo types, which denote those flip-flops that are not provided by the library. For example, assume that a library only supports two kinds of flip-flops whose bit widths are 1 and 4, respectively.

In order to use a binary tree to denote a combination whose bit width is 4, there must exist flip-flops whose bit widths are 2 and 3 in L [please see the last two binary trees in Fig. 9(e) for example].the combination table has been created by creating two pseudotypes. For example, suppose a library L only provides two types of flip-flops, whose bit widths are 1 and 4 (i.e., bmin = 1 and bmax = 4), in Fig. 9(a). We first initialize two combinations n1 and n2 to represent these two types of flip-flops in the table T [see Fig. 9(a)]. Next, the function Insert Pseudo Type is performed to check whether the flip-flop types with bit widths between 1 and 4 exist or not. Thus, two kinds of flip-flop types whose bit widths are 2 and 3 are added into L, and all types of flip-flops in L are sorted according to their bit widths [see Fig. 9(b)]. Now, for each combination in T, we would build a binary tree with 0-level, and the root of the binary tree denotes the combination. Next, we try to build new legal combinations according to the present combinations. By combing two 1-bit flip-flops in the first combination, a new combination n3 can be obtained [see Fig. 9(c)]. Similarly, we can get a new combination n4 (n5) by combining n1 and n3(two n3"s) [see Fig. 9(d)]. Finally, n6 is obtained by combing n1 and n4. All possible combinations of flip-flops are shown in Fig. 9(e). Among these

combinations, n5 and n6 are duplicated since they both represent the same condition, which replaces four 1-bit flip-flops by a 4-bit flip-flop. To speed up our program, n6 is deleted from T rather than n5 because its height is larger. After this procedure, n4 becomes an unused combination [see Fig. 9(e)] since the root of binary tree of n4 corresponds to the pseudo type, type3, in L and it is only included in n6. After deleting n6, n4 is also need to be deleted.

The last combination table T is shown in Fig. 9(f). In order to enumerate all possible combinations in the combination table, all the flip-flops whose bit widths range between b max and b min and do not exist in L should be inserted into L in the above procedure. However, this is time consuming. To improve the running time, only some types of flip-flops need to be inserted. There exist several choices if we want to build a binary tree corresponding to a type j. However, the complete binary tree has the smallest height. Thus, for building a binary tree of a certain combination ni whose type is type j , only the flip-flops whose bit widths are (_b(type j)/2_) and (b(type j)–_b(type j)/2_) should exist in L. New method for inserting flip-flop in two pseudo type has been carried out In the new procedure, it first adds two pseudo types of flip-flops whose bit widths are 3 and 4, respectively, for the flip-flop with 7-bit (i.e., L becomes [1 3 4 7]). Next, the flip-flop whose bit width is 2 is added to L for the flip-flop with 4-bit (i.e., L becomes [1 2 3 4 7]). For the flip-flop with 3-bit, the procedure stops because flop-flops with 1 and 2 bits already exist in L. In the new procedure, we do not need to insert 5- and 6-bit pseudo types to L.



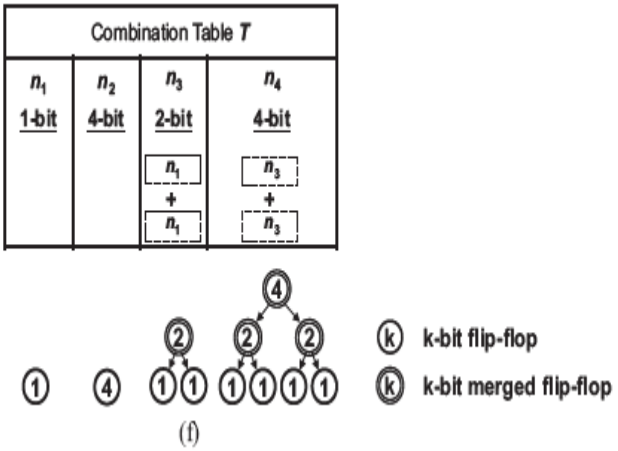


Fig. 7. Example of building the combination table. (a) Initialize the library L and the combination table T . (b) Pseudo types are added into L, and the corresponding binary tree is also build for each combination in T. (c) New combination $n3$ is obtained from combining two $n1$s. (d) New combination $n4$ is obtained from combining $n1$ and $n3$, and the combination $n5$ is obtained from combining two $n3$s. (e) New combination $n6$ is obtained from combining $n1$ and $n4$. (f) Last combination table is obtained after deleting the unused combination in (e).

4.2.6 Merge Flip-Flops

We have shown how to build a combination table in Section III-B. Now, we would like to show how to use the combination table to combine flip-flops in this subsection. To reduce the complexity, we first divide the whole placement region into several sub regions, and use the combination table to replace flip-flops in each sub region.

Then, several sub regions are combined into a larger sub region and the flip-flops are replaced again so that those flip-flops in the neighboring sub regions can be replaced further. Finally, those flip-flops with pseudo types are deleted in the last stage because they are not provided by the supported library. Fig. 10 shows this flow.

1) Region Partition (Optional): To speed up our problem, we divide the whole chip into several sub regions. By suitable partition, the computation complexity of merging flip-flops can be reduced significantly (the related quantitative analysis will be shown in Section V). As shown in Fig. 11, we divide the region into several sub regions, and each sub region contains six bins, where a bin is the smallest unit of a sub region.

2) Replacement of Flip-flops in Each Sub region: Before illustrating our procedure to merge flip-flops, we first give an equation to measure the quality if two flip-flops are going to be replaced by a new flip-flop as follows:

$$\text{Cost} = \text{routing length} - \alpha \times \sqrt{\text{available\_area}} \qquad (5)$$

Where routing_length denotes the total routing length between the new flip-flop and the pins connected to it, and available_ area represents the available area in the feasible region for placing the new flip-flop. α is a weighting factor

(the related analysis of the value α will be shown in Section V). The cost function includes the term routing_length to favor a replacement that induces shorter wire length. Besides, if the region has larger available space to place a new flip-flop, it implies that it has higher opportunities to combine with other flip-flops in the future and more power reduction. Thus, we will give it a smaller cost. Once the flip-flops cannot be merged to a higher-bit type (as the 4-bit combination *n*4 in Fig. 9), we ignore the *available area* in the cost function, and hence α is set to 0. After a combination has been built, we will do the replacements of flip-flops according to the combination table.

First, we link flip-flops below the combinations corresponding to their types in the library. Then, for each combination ***n*** in ***T*,** we serially merge the flip-flops linked below the left child and the right child of ***n*** from leaves to root. Based on its binary tree, we can find the combinations associated with the left child and right child of the root. Hence, the flip-flops in the lists, named *l*eft and *l*right, linked below the combinations of its left child and its right child are checked (see Lines 2 and 3). Then, for each flip-flop ***f i*** in *l*eft, the best flip-flop *b*est in *r*ight, which is the flip-flop that can be merged with ***f i*** with the smallest cost recorded in *c*best, is picked. For each pair of flip-flops in the respective list, the combination cost [based on (5)] is computed if they can be merged and the pair with the smallest cost is chosen .finally, we add a new flip-flop ***f'*** in the list of the combination *n* and remove the picked flip-flops which constitutes the ***f*** „For example, given a library containing three types of flip-flops (1-, 2-, and 4-bit), we first build a combination table ***T*** as shown in Fig. 12(a). In the beginning, the flip-flops with various types are, respectively, linked below *n*1, *n*2, and *n*3 in ***T*** according to their types. Suppose we want to form a flip flop in *n*4, which needs two 1-bit flip-flops according to the combination table.

Each pair of flip-flops in *n*1 are selected and checked to see if they can be combined (note that they also have to satisfy the timing and capacity constraints described in Section II). If there are several possible choices, the pair with the smallest cost value is chosen to break the tie. In Fig. 12(a), *f*1 and *f*2 are chosen because their combination gains the smallest cost. Thus, we add a new node *f*3 in the list below *n*4, and then delete *f*1 and *f*2 from their original list [see Fig. 12(b)]. Similarly, *f*4 and *f*5 are combined to obtain a new flip-flop *f*6, and the result is shown in Fig. 12(c). After all flip-flops in the combinations of 1-level trees (*n*4 and *n*5*)* are obtained as shown in Fig. 12(d), we start to form the flip-flops in the combinations of 2-level trees (*n*6, and *n*7*)*. In Fig. 12(e), there exist some flip-flops in the lists below *n*2 and *n*4, and we will merge them to get flip-flops in *n*6 and *n*7, respectively. Suppose there is no overlap region between the couple of flip-flops in *n*2 and *n*4. It fails to form a 4-bit flip-flop in *n*6. Since the 2-bit flip-flops *f*3 and *f*6 are merge able, we can combine them to obtain a 4-bit flip-flop *f*10 in *n*7.

Finally, because there exists no couple of flip-flops that can be combined further, the procedure finishes as shown in Fig. 12(f). If the available overlap region of two flip-flops exists, we can assign a new one to replace those flip-flops. Once there is sufficient space to place the new flip-flop in the available region, the algorithm will perform the replacement, and the new generated flip-flop will be placed in the grid that makes the wire length between the flip-flop and its connected pins smallest. If the capacity constraint of the bin, *Bk*, which the grid belongs to will be violated after the new flip-flop is placed on that grid, we will search the bins near *Bk* to find a new available grid for the new flip-flop. If none of bins which are overlapped with the available region of new flip-flop can satisfy the capacity constraint after the placement of new flip-flop, the program will stop the replacement of the two flip-flops.

3) Bottom-Up Flow of Sub region Combinations (Optional): As shown in Fig. 13(a), there may exist some flip-flops in the boundary of each sub region that cannot be replaced by any flip-flop in its sub region. However, these flip-flops may be merged with other flip-flops in neighboring sub regions as shown in Fig. 13(b). Hence, to reduce power consumption furthermore, we can combine several sub regions to obtain a larger sub region and perform the replacement again in the new sub region again.

The procedure repeats until we cannot achieve any replacement in the new sub region. Fig. 14 gives an example for this hierarchical flow. As shown in Fig. 14(a), suppose we divide a chip into 16 sub regions in the beginning. After the replacement of flip-flops is finished in each sub region, four sub regions are combined to get a larger one as shown in Fig. 14(b). Suppose some flip-flops in new sub regions still can be replaced by new flip-flops in other new sub regions, we would combine four sub regions in Fig. 14(b) to get a larger one as shown in Fig. 14(c) and perform the replacement in the new sub region again.

As the procedure repeats in a higher level, the number of merge able flip-flops gets fewer. However, it would spend much time to get little improvement for power saving. To consider this issue, there exists a trade-off between power saving and time consuming
in our program.

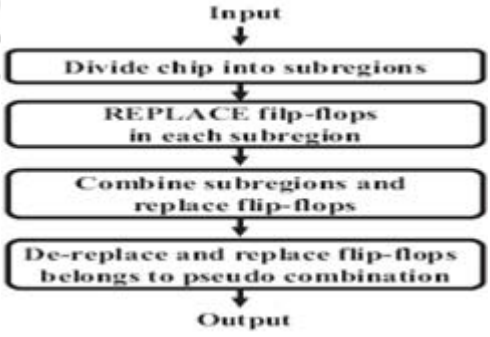


Figure 10: Detailed flow to merge flip-flops.

For example, if there still exists a flip-flop, *fi* , belonging to *n*3 after replacements in Fig. 9(f), we have to de-replace *fi* into two flip-flops originally belongs to *n*1. After de-replacing, we will do the replacements of flip-flops according to ***T*** without consideration of the combinations whose corresponding type is pseudo in ***L***.

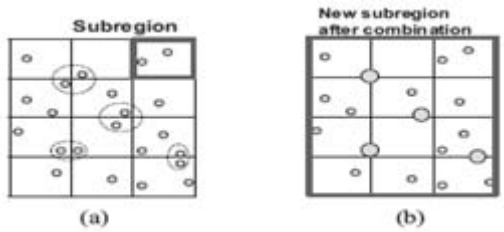


Figure 13: Combination of flip-flops near sub region boundaries. (a) Result of replace flip-flops in each sub region. (b) Result of replace flip-flops in each new sub region which is obtained from combining twelve sub region in (a).

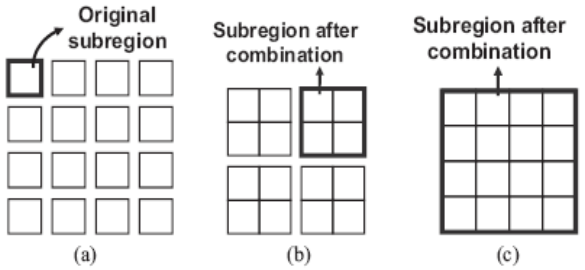


Figure 14: Combination of sub regions to a larger one. (a) Placement is originally partitioned into 16 sub regions for replacement. (b) Sub region bounded by bold line is obtained from combining four neighboring sub regions in (a). (c) Sub region bounded by bold line is obtained from combining four sub regions in (b).

4.2.7 Boolean Expressions

Boolean Expression: Combining the variables and operation yields Boolean expressions. Boolean Function: A Boolean function typically has one or more input values and yields **a result**, based on these input value, in the range {0, 1}.

A Boolean operator can be completely described using a **table** that list inputs, all possible values for these inputs, and the resulting values of the operation. A **truth table** shows the **relationship**, in tabular form, between the input values and the result of a specific Boolean operator or function on the input variables.

The AND operator is also known as a **Boolean product**. The Boolean expression xy is equivalent to the expression x * y and is read "x and y." The behavior of this operator is characterized by the truth table.

4.2.8 Boolean Identities

Boolean expression can be simplified, but we need new **identities**, or **laws**, that apply to Boolean algebra instead of regular algebra.

Table 5: Basic Identities of Boolean Algebra

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0+x = x$
Null (or Dominance) Law	$0x = 0$	$1+x = 1$
Idempotent Law	$xx = x$	$x+x = x$
Inverse Law	$x\bar{x} = 0$	$x+\bar{x} = 1$
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$
Absorption Law	$x(x+y) = x$	$x+xy = x$
DeMorgan's Law	$(\overline{xy}) = \bar{x}+\bar{y}$	$(\overline{x+y}) = \bar{x}\bar{y}$
Double Complement Law	$\bar{\bar{x}} = x$	

4.2.9 Flip-Flops

Many people use the terms **latch** and flip-flop interchangeably. Technically, a latch is level triggered, whereas a flip-flop is edge triggered.

In order to "remember" a past state, sequential circuits rely on a concept called **feedback**. This simply means the output of a circuit is fed back as an input to the same circuit. A more useful feedback circuit is not composed of two NOR do gates resulting in the most basic memory unit call an **SR flip-flop**. SR stands for "**set/reset**."
Q(t) means the value of the output at time t. Q(t+1) is the value of Q after the **next** clock pulse.

When both S and R are 1, the SR flip-flop is **unstable**.

5. Conclusion

This is evidence in section 7 and 8 where the Flip Flop Extensions at 87.5% active states utilization is designed with one gate less than the conventional SR-Flip Flop. The uniqueness of this study is that computer memory speed performance can be enhanced through conventional SR-FF modification just as it is currently being done with its processor counterpart. This is a great advantage in performance over the conventional Flip Flops because fewer gates enhance performance (i.e., gate delay represents performance). The Flip Flop extension memory cell is also portable (less transistors) and cheaper because it requires fewer transistors as against the conventional Flip Flops. An important issue in digital device design is that numbers of transistors represent hardware cost because in essence, maximizing performance and minimizing cost in digital devices are part of the factors in seeking alternative design on more efficient and effective Flip flops. Efforts should be geared towards verifying the effectiveness and efficiency of these newly design Flip Flops Extension over the existing conventional Flip Flops.

References

[1] J. P. Abraham and S. Mathew, An Attempt to Improve the Processor Performance by Proper Memory Management for Branch Handling, IJCSEA, 2013, Vol.3, No.4
[2] F. Hamzaoglu, Y. Te, A. Keshavarzi, and K. Zhang, Dual Vt-SRAM cells with full-swing single-ended bit

line sensing for high-performance on-chip cache in 0.13µm technology generation, International Symposium on Low Power Electronics and Design, pp. 15–19, 2000
[3] J. Inouye, P Molloy and M. Wisler, Overcoming the Memory Wall, Oregon State University, 2012
[4] L. Jamal, Sharmin, A. Mottalib, H. Babu, Design and Minimization of Reversible Circuits for a Data Acquisition and Storage System, IJET, 2012, Vol. 2
[5] Jyoti, M. R. Tripathy and Vijeta, Comparison of Conditional Internal Activity Techniques for Low Power Consumption and High Performance Flip-Flops, International Journal of Computer Science and Telecommunications, ISSN 2047-3338, 2012, Volume 3, Issue 2
[6] T. Kavitha and V. Sumalatha, A new Reduced Clock Power Flip Flop for future System On-Chip (SOC) Applications, IJCTT, 2012, Vol. 3.
[7] C. Kim and K. Roy, Dynamic Vt SRAM: a leakage tolerant cache memory for low voltage microprocessor, in Proc. of International Symposium on Low Power Electronics and Design, pp. 251-254, 2002
[8] J. P. Abraham and S. Mathew, An Attempt to Improve the Processor Performance by Proper Memory Management for Branch Handling, IJCSEA, 2013, Vol.3, No.4
[9] F. Hamzaoglu, Y. Te, A. Keshavarzi, and K. Zhang, Dual Vt-SRAM cells with full-swing single-ended bit line sensing for high-performance on-chip cache in 0.13µm technology generation, International Symposium on Low Power Electronics and Design, pp. 15–19, 2000
[10] J. Inouye, P Molloy and M. Wisler, Overcoming the Memory Wall, Oregon State University, 2012
[11] L. Jamal, Sharmin, A. Mottalib, H. Babu, Design and Minimization of Reversible Circuits for a Data Acquisition and Storage System, IJET, 2012, Vol. 2
[12] Jyoti, M. R. Tripathy and Vijeta, Comparison of Conditional Internal Activity Techniques for Low Power Consumption and High Performance Flip-Flops, International Journal of Computer Science and Telecommunications, ISSN 2047-3338, 2012, Volume 3, Issue 2
[13] T. Kavitha and V. Sumalatha, A new Reduced Clock Power Flip Flop for future System On-Chip (SOC) Applications, IJCTT, 2012, Vol. 3.
[14] C. Kim and K. Roy, Dynamic Vt SRAM: a leakage tolerant cache memory for low voltage microprocessor, in Proc. of International Symposium on Low Power Electronics and Design, pp. 251-254, 2002.
[15] A.S.Syed Navaz, C.Prabhadevi & V.Sangeetha"Data Grid Concepts for Data Security in Distributed Computing" January 2013, International Journal of Computer Applications, Vol 61 – No 13, pp 6-11.
[16] A.S.Syed Navaz, H.Iyyappa Narayanan & R.Vinoth." Security Protocol Review Method Analyzer (SPRMAN)", August – 2013, International Journal of Advanced Studies in Computers, Science and Engineering, Vol No – 2, Issue No – 4, pp. 53-58.

Author Profile

M.Valli received M.Sc in Information Technology and M.Phil in Computer Science. Currently she is working as an Asst.Professor at St.Joseph College of Arts & Science (Autonomous), Cuddalore. India. Her areas of interest are Network Security, Computer Graphics & Data mining.

Dr. R. Periyasamyi received M.Sc., M.Phil., PGDCA., PhD. Currently she is working as an Associate Professor at Nehru Memorial College (Autonomous), Trichy. India. His areas of interest are Data mining & Neural Networks.