International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:3, 2007

A Set Theory Based Factoring Technique and Its Use for Low Power Logic Design

Padmanabhan Balasubramanian, and Ryuta Arisaka

***Abstract*—** Factoring Boolean functions is one of the basic operations in algorithmic logic synthesis. A novel algebraic factorization heuristic for single-output combinatorial logic functions is presented in this paper and is developed based on the set theory paradigm. The impact of factoring is analyzed mainly from a low power design perspective for standard cell based digital designs in this paper. The physical implementation of a number of MCNC/IWLS combinational benchmark functions and sub-functions are compared before and after factoring, based on a simple technology mapping procedure utilizing only standard gate primitives (readily available as standard cells in a technology library) and not cells corresponding to optimized complex logic. The power results were obtained at the gate-level by means of an industry-standard power analysis tool from Synopsys, targeting a 130nm (0.13µm) UMC CMOS library, for the typical case. The wire-loads were inserted automatically and the simulations were performed with maximum input activity. The gate-level simulations demonstrate the advantage of the proposed factoring technique in comparison with other existing methods from a low power perspective, for arbitrary examples. Though the benchmarks experimentation reports mixed results, the mean savings in total power and dynamic power for the factored solution over a non-factored solution were 6.11% and 5.85% respectively. In terms of leakage power, the average savings for the factored forms was significant to the tune of 23.48%. The factored solution is expected to better its non-factored counterpart in terms of the power-delay product as it is well-known that factoring, in general, yields a delay-efficient multi-level solution.

***Keywords*—**Factorization, Set theory, Logic function, Standard cell based design, Low power.

I. Introduction

TWO-LEVEL circuits are widely used to implement Boolean functions through conventional programmable logic devices such as PLA or PAL. However, in modern VLSI design, they are economically implemented by multilevel circuits. Factorization techniques are key tools in facilitating multilevel synthesis. Finding a minimum factored expression can be a cumbersome task. So we are compelled to resort to heuristic algorithms in order to find a 'good', if not best solution in reasonable time. The general strategy employed by such algorithms is as follows. A divisor of a Boolean function F to be factored is singled out. Then F is divided by this divisor. Such a procedure is recursively applied to the quotient and the remainder of the division. Computing time and factorization quality depend on the way in which a divisor is chosen and the type of division performed i.e., algebraic or Boolean. In this paper, a novel heuristic developed on the basis of set theory for algebraic factorization of single-output Boolean functions is presented. The algorithm has been implemented as a stand-alone factorization tool in a high-level language, Java in MS-Windows OS, and takes as input the reduced logic expressions resulting from a standard two-level logic minimizer such as Espresso and outputs the factorized solution in the same algebraic expression format after compilation. Simultaneous factorization of any number of different single output Boolean functions is made possible. Rather than comparing the time taken for factorization with other existing techniques, we instead focus on the power quality of the resulting solution after implementation using standard cells corresponding to a 130nm UMC CMOS library.

The proliferation of portable hand-held electronics combined with increasing packaging costs is forcing circuit designers to adopt low power design methodologies. Power wall is a clear roadblock in the semiconductor industry [4]. Low power design of ASICs result in increased battery life and enhances reliability. Infact, the Semiconductor Industry Association technology roadmap [5] has underlined low power design techniques as a critical need. Hence, it is indispensable for circuit designers to acknowledge the importance of limiting power consumption and subsequently improve energy efficiency, possibly at all levels of the design hierarchy, starting from even the lower levels of design abstraction. Gate-level optimization achieves power savings; in some specific cases more than 50% reduction in power, without loss of performance, may be achieved [2]; though in general the reduction is around 5%-15% [3]. The other advantage being that logic-level optimization is relatively low cost in terms of design effort in comparison with strategies employed at other levels. We analyze the effect of factoring of logic functions from a low power point of view and make an effort to address the burning issue of power dissipation at the gate-level mainly with respect to the above operation.

The remaining part of this paper is organized as follows. In section 2, the different components of power consumption in digital CMOS circuits are discussed. In the next section, basic and background information pertaining to logic functions has been first presented. Then the issue of factorization, which is

Padmanabhan Balasubramanian is with the School of Computer Science, The University of Manchester, Manchester, MAN M13 9PL, UK (phone: +44-161-275 6294; e-mail: padmanab@cs.man.ac.uk).
Ryuta Arisaka is with the School of Computer Science, The University of Manchester, Manchester, MAN M13 9PL, UK (phone: +44-161-275 6294; e-mail: ryuta@cs.man.ac.uk).

basically a logic optimization (restructuring step), performed with the intention of reducing the literal costs at the technology independent stage is briefly dealt with. Algebraic and Boolean division operations are concisely explained with simple Boolean functions. In section 4, the proposed factorization technique based on the notion of set theory is then presented with an illustration, followed by a generalized algorithm. Section 5 elucidates the hardware implementation issues involved and about the choice of the base-function set for technology binding. In the subsequent section, a variety of examples follow; to illustrate the significance of the proposed factorization technique in comparison with other schemes, from a low power perspective. In section 7, the power estimation methodology is highlighted. Then the impact of the factoring operation on logic functions is then analyzed extensively from a power dissipation perspective for many MCNC/IWLS combinational benchmark functions and sub-functions [7] [8]. The power results were obtained using an industry-standard power analysis tool, Synopsys PrimePower, and correspond to a 130nm (0.13µm) UMC CMOS library, comprising high density standard cells, for the typical case. The wire loads were automatically selected for simulation purpose, based on the cells used and their assigned drive strengths. Section 8 presents a concise summary and also the conclusion borne out of this research, followed by the bibliography.

II. Power consumption in digital CMOS circuits

CMOS has long been considered the technology of choice for low power applications. The continuous shrinking of feature sizes has made it possible to achieve even greater integration of complex functions on a single chip. However, the higher chip densities have resulted in one to two orders of magnitude increase in the power consumption of many higher-end processors. The point is being rapidly reached when reduction of power dissipation becomes a most important hurdle that designers and manufacturers need to tackle.

Power consumption in CMOS circuits falls into two broad categories: dynamic power ($P_{dynamic}$) and static power (P_{static}). Dynamic power is the power dissipated when the circuit is active. It is composed of two kinds of power viz. switching power ($P_{switching}$) and internal power ($P_{internal}$). $P_{switching}$ is due to the charging and discharging of load capacitance at the output of the cell as it makes transitions between '0' and '1'. The total load capacitance at the output of a driving cell is modeled as the sum of interconnect and gate capacitances on the driving output. It is typically expressed as $C_L V_{dd}^2 E(t)$, where C_L is the load capacitance, V_{dd} is the supply voltage and $E(t)$ is the expected number of times that the gate switches, also called transition activity. The quadratic dependence of $P_{switching}$ on V_{dd} indicates that scaling down the supply voltage will have the greatest impact on reducing $P_{switching}$. This also avoids hot-carrier effects in short-channel devices. However, the threshold voltage V_t also has to be scaled down because otherwise it has a much greater detrimental impact on the delay when small geometry devices are used [18]. Thus scaling V_t by the same factor as V_{dd} is needed so as not to adversely impact delay. However, reducing V_t in small geometry MOSFETs results in an exponential increase in the standby current [19]. $P_{internal}$ is any power dissipated within the boundary of a cell. During switching, a circuit dissipates internal power by the charging or discharging of any existing capacitances that are internal to the cell (also called intrinsic capacitances). $P_{internal}$ also includes power dissipated during a momentary short circuit between the pull-up and pull-down networks of a standard cell, called as short-circuit power, P_{sc}. Static power consumption (P_{static}) is the power dissipated by a gate when it is not switching, i.e., when it is inactive. The main component of static power results from source-to-drain subthreshold leakage, which is caused by reduced threshold voltages that prevent the gate from completely turning off. In other words, it is mainly due to the leakage current caused by the reverse-biased junction leakage and sub-threshold leakage (devices that conduct while in the OFF-state – subthreshold conduction). Since power is dissipated when current leaks between the diffusion layers and the substrate, static power is also referred to as leakage power. Simulation results given in [20] show that the power dissipation due to the standby current dominates the switching power at low threshold voltages. Predictions on future technologies project that the leakage power will be so high that it will become substantial even when the chip is in active mode.

III. Factorization of logic functions

A. Preliminaries

In this section, some background information about Boolean function, network and the terminologies related with logical division are first stated.

1) Definition 1: Boolean function

A single output Boolean function is a mapping from n-dimensional ($n \geq 0$) Boolean space into a 1-dimensional one: $\{0,1\}^n \rightarrow \{0,1,d\}$, where '$d$' denotes a don't care condition. If this condition does not exist, then the function is a completely specified function (CSF). Each of the 2^n nodes in the Boolean space corresponds to a minterm. If a minterm is mapped to output 1 (0 or d), then it is called an ON-set (OFF-set or DC-set) minterm.

2) Definition 2: Boolean network

A Boolean network is usually modeled as a directed acyclic graph (DAG) with *nodes* represented by Boolean functions. A DAG is distinguished from a tree structure in that its nodes can have unlimited fan-in and unlimited fan-out. The sources of the graph are the primary inputs of the network; the sinks are the primary outputs. The inputs of a node are called its *fan-in's*. The output of a node may be an input to other nodes called its *fan-out's*. An *edge* connects two nodes that are in the *fan-in*/*fan-out* relationship.

3) Definition 3: Kernel and Co-Kernel

The quotient resulting from the algebraic division of an expression, F, by a cube c (i.e., F/c) is the *kernel* k of F, if

there are at least two cubes in the quotient and the cubes do not have any common literal. The cube divisor *c* used to obtain the kernel is called its *co-kernel*. Different co-kernels may produce the same kernel: hence, the co-kernel of a kernel is not unique. If a kernel has no kernels except itself, it is said to be a level-0 kernel. A kernel is said to be of level *n* if it has at least one level-*(n-1)* kernel but no kernel, except itself, of level *n* or greater.

Let us consider the following Boolean expression, given by,

$$Z(a,b,c,d,e,f,g) = abc + ac'g + b'df + b'cde \quad (1)$$

The quotient of *Z* and the cube *a* is then,

$$Z/a = bc + c'g \quad (2)$$

Similarly the quotient of *Z* and the cube *b'* is

$$Z/b' = df + cde \quad (3)$$

Z/a is a kernel of *Z*, since it has two cubes and no common literal. The co-kernel is *a*. However, *Z/b'* is not a kernel of *Z*, since literal *d* is common to both cubes of *Z/b'*.

B. Factorization – Algebraic and Boolean division

Factoring Boolean functions is one of the basic operations in algorithmic logic synthesis. The objective of factorization is to represent a Boolean function in a logically equivalent factored form but with a minimum number of literals. An optimal (shortest-length) factorized solution for an arbitrary Boolean function is a problem which cannot be solved in polynomial time; in many situations (NP-hard), and so all practical algorithms for factoring are heuristic and provide a correct, logically equivalent formula, but not necessarily a minimal length solution in each case. This type of optimization step will yield a minimum area for the physical realization of this function. Algebraic algorithms for factorization have been developed previously [10] [11] and are widely used in commercial environments due to their speed. On the other hand, Boolean factoring [12] [13] is not widely used because of its computational complexity even though it gives better results in many cases. The main difficulty in the latter being the difficulty to easily figure out good candidate divisors for a function, which is not usually straight-forward.

Factoring is the translation of a function in the sum-of-products form (also called disjunctive form) to a form with parentheses and having a minimum number of literals [10]. For e.g. *a*, *ab'c'*, *a(b+c+d)+e,* are all factored forms. Thus it is equivalent to a parenthesized algebraic expression and is most appropriate one for use in multilevel logic synthesis. A factored form is isomorphic to a tree structure, where each internal node is an AND or OR operator and each leaf is a literal. There are mainly two methods to obtain the factored form of a two-level representation of the function: Algebraic division, also known as weak division which is quite fast and Boolean division, also known as strong division which is slower but capable of giving better results in many cases. In general, the algebraic methods are fast because the logic function is treated as a polynomial, and hence fast methods of manipulation are available. Boolean factoring is generally non-polynomial, and there is not much information about the implementation of such algorithmic procedures.

Let us assume two Boolean expressions, *f* and *g*. If there is an operation which generates expressions *h* and *r* such that $f = gh + r$, where *gh* is an algebraic product (i.e. *g* and *h* have no common variable), then this operation is called an algebraic division. For example, if $f = abd + bcd + a'c + b'd'$ and $g = a + c$, the algebraic (polynomial) division will yield

$$f = gh + r = bd(a + c) + a'c + b'd' \quad (4)$$

Another form of division used in factoring logic expressions uses the identities of Boolean algebra (for e.g. $yy' = 0$, $yy = y$, and $y + y' = 1$ for a variable *y*). Thus, for an expression, $f = pq + t$, *pq* is a Boolean product [i.e. *p* and *q* have one or more common variable(s)], then the division of *f* by *p* is called a Boolean division. Hence Boolean division for the original expression of *f* will result in the following factored form.

$$f = pq + t = (bd + a')(a + c) + b'd' \quad (5)$$

IV. Set Theory Based Algebraic Factoring Technique

Some of the widely used algebraic factorization methods are usually found embedded in open-source multi-level logic synthesis systems such as MIS [15], SIS [14] or in commercial environments. This necessitates the need for a stand-alone factorizer which could then be modified to suit different requirements at a later stage. Before proceeding with the listing of the proposed factoring heuristic, let us familiarize ourselves with the terminology defined for a Boolean cube, *c*: the *description set* of a cube, *D(c)*. *D(c)* specifies the set of all literals in their actual form, which a particular Boolean cube *c* is dependent upon for its evaluation to a logic value of '1'. For e.g. if $F = a'bd + b'cd'$, where cubes, $C_1 = a'bd$ and $C_2 = b'cd'$, then $D(C_1) = \{a',b,d\}$ and $D(C_2) = \{b',c,d'\}$ and so the set intersection of the two cubes, $D(C_1) \cap D(C_2) = \Phi$, with its cardinality given by $| D(C_1) \cap D(C_2) | = 0$.

The set theory based factorizing technique, which also treats a logic expression as a polynomial is described through steps 1 to 11 of the proposed algorithm listed below.

A. Algorithm

Given a logic function *F*:

1) Minimize *F* into two-level logic to obtain *F** (where *F* and *F** are logically equivalent)
2) For each cube, *c* in *F**, define its *D(c)*
3) Perform set intersection of *D(c)* with the description set of each and every cube in *F** independently
4) Enumerate the cardinality of all the set intersection operations
5) Choose those intersection operations which yield the highest cardinality

6) Extract the shared literal(s) from the cube(s) which correspond to highest cardinality
7) The shared literal(s) now correspond to the *co-kernel*
8) Remaining literals in each of those cubes (logically OR-ed) form the *kernel*
9) Check the cubes grouped in *F**
10) For the remaining unchecked cubes in *F**, repeat *steps 2 to 8* (even with existing co-kernel(s)) till all cubes are checked
11) The resulting solution is identified as *G*, factored form of *F**
12) (*Extract* and *Group* operations) – Parse *G* to find whether similar kernels exist independently; otherwise go to step 15
13) If so, the kernel is extracted once and its respective co-kernels are logically OR-ed
14) Recursively execute *steps 12 and 13* till no more similar, but independent kernels are found
15) The final algebraically factorized solution for the given Boolean function is obtained

In short, the above algorithm is described as follows. Largest single cube common divisors are first extracted from a Boolean function specified in minimized disjunctive form, based on the cardinality of the intersection operation between the description sets of two distinct cubes considered at a time. This procedure is then iterated until no more single cube divisors can be isolated in the function. This completes the algebraic factoring step. The resulting kernels, if exist independently, and are also found to be shared between different co-kernels are then extracted and grouped according to the distributive axiom. The final solution is not only factored but also does not contain any logic duplication.

V. Hardware Realization Issues and Choice of Base-Function Set

In this section, information about the issues involved in the physical implementation of the logic expressions is discussed. This is important in this context that the minimized two-level solutions output by standard tools such as Espresso [4] cannot be implemented as it is (cannot be synthesized) due to the fan-in restrictions imposed on the gates available in a physical standard cell library. So it is clear that there is a need for technology binding here. Technology binding is the process of mapping (implementing) a technology independent description in a particular technology [16]. The role of technology mapping, as seen here, is not to change the structure of the circuit (as this will amount to modifying the actual synthesis solution), for e.g. by finding common sub-expressions between two or more parts of the global function; but to finish the synthesis of the circuit by performing the final gate selection from a particular library. The actual role played by technology mapping here is the choice of gate primitives belonging to the cell library in order to implement the logic equations. When implementing large logic function terms, there arises a need to partition the input field by decomposing, so that it can be implemented as a combination of sub-function terms. It is not always obvious how best to achieve this. So the problem involves selecting the most judicious input variable sub-sets so that the overall term may be implemented by a suitable combination of sub-terms. We consider this issue here mainly from a low power point of view. We do this so as to specifically study the gain secured by factorization when combined along with technology mapping in comparison with pure technology mapping. So for the present, we do not introduce other logic transformations for optimization, since, we are more interested in the issue of technology binding after local optimization.

The choice of a base-function set is at the heart of any technology mapping algorithm. Also, the choice of a set of base-functions could be arbitrary as long as it is functionally complete [16]. The goal here is to find that base-function set which would provide the highest level of optimization (mainly power optimization) with a small set of patterns. According to [16], the granularity of a base-function set affects the optimization potential. With this approach, the logic function, *Y = (efgh + ijkl + mnop + qrst)* requires only one pattern for realization in NAND-NAND logic style – a tree of five four-input NAND gates; with a base-function set comprising two-input, three-input and four-input NAND gates and inverting buffers. Representing all patterns for this same function using two-input NAND gates and inverters would require eighteen patterns. So a finer resolution base-function set would allow for more covers, and hence better quality solutions. In our case, we consider all individual gate primitives of a standard cell library to constitute the base-function set. The above discussion is further clarified with the following examples for AND-OR-Invert (AOI) logic format realization.

For a cube, *a'b'c'd'e'f'*, let us consider its implementation via, three different tree structures by means of technology mapping: using only 2-input AND-gates and inverters (*imp1*), using 2-input, 3-input AND gates and inverters (*imp2*) and by a maximum fan-in based mapping using 3-input, 4-input AND gates and inverters (*imp3*); represented by figures 1, 2 and 3.

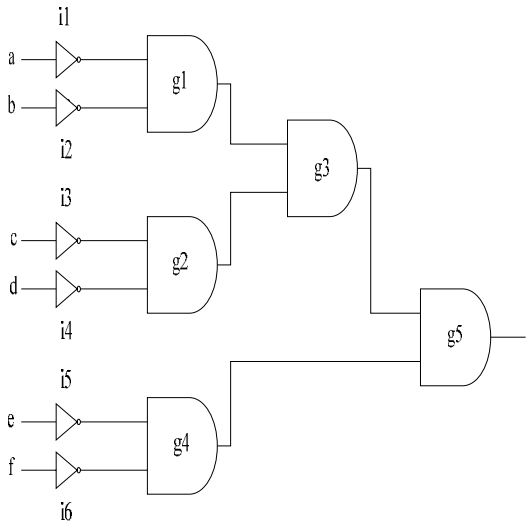

Fig. 1 Technology mapping using 2-input cells and inverters (*imp1*)

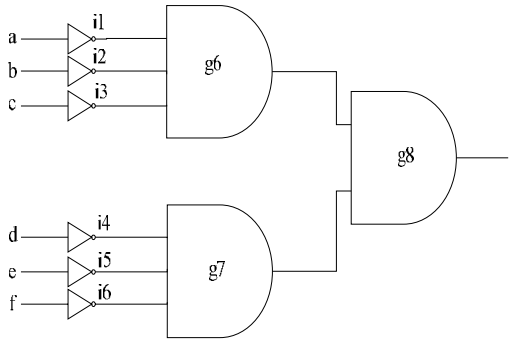

Fig. 2 Technology binding with 2, 3-input cells and inverters (*imp 2*)

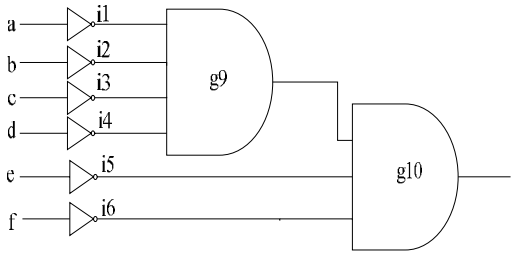

Fig. 3 Technology binding based on maximum fan-in based implementation (*imp3*)

TABLE I
WORST-CASE AVERAGE DELAY COMPARISON

Implementation style	Critical path	Worst-case average delay (ns)
imp1	i2-g2-g3-g5	0.17125
imp2	i2-g7-g8	0.132
imp3	i2-g9-g10	0.156

The above table gives the maximum average delay computed along the respective critical paths of the above three implementations. The average delay, estimated as the mean of the low-to-high (rise) and high-to-low (fall) delays encountered by the signal while traversing through logic gates, is accurately determined at the gate level by the timing analyzer, PrimeTime for a 130nm UMC CMOS process. The wire loads were back-annotated by the tool automatically before performing timing analysis.

TABLE II
POWER DISSIPATION OF DIFFERENT IMPLEMENTATIONS
(FOR TYPICAL CASE: SUPPLY = 1.2V, TEMPERATURE = 25°C)

Implementation style	Power dissipation components		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
imp1	3.26914	3.23945	29.6877
imp2	2.35462	2.33404	20.5844
imp3	1.87136	1.85511	16.2458

The last implementation style leads to lesser power consumption than the other two, as can be seen from Table 2, for a typical case library specification. This is mainly because of a reduction in the number of cell instances, from a simulation point of view. However, the reason for this phenomenon is captured more accurately at the device level in [17]. To verify this, the simulation has been extended targeting best case and worst case library specifications as well and they are found to be in good agreement with the above, as evident from Tables 3 and 4. The power results were obtained for a clock frequency of 100MHz. Table 5 further shows that the technology binding procedure identified as *imp3* betters the other two in terms of the power-delay product (PDP) as well, for all the three target library scenarios.

TABLE III
POWER DISSIPATION OF DIFFERENT IMPLEMENTATIONS
(FOR BEST CASE: SUPPLY = 1.32V, TEMPERATURE = 0°C)

Implementation style	Power dissipation components		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
imp1	4.25175	4.16056	91.1835
imp2	3.04785	2.98433	63.5213
imp3	2.40974	2.35994	49.8044

TABLE IV
POWER DISSIPATION OF DIFFERENT IMPLEMENTATIONS
(FOR WORST CASE: SUPPLY = 1.08V, TEMPERATURE = 125°C)

Implementation style	Power dissipation components		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
imp1	2.63746	2.56383	73.6291
imp2	1.91433	1.86302	51.3116
imp3	1.52619	1.48606	40.1256

TABLE V
PDP EVALUATION FOR DIFFERENT IMPLEMENTATIONS

Implementation style	Typical case (fJ)	Best case (fJ)	Worst case (fJ)
imp1	0.5598	0.7281	0.4517
imp2	0.3108	0.4023	0.2527
imp3	0.2919	0.3759	0.2381

To confirm the veracity of the above argument, a benchmark sub-function of eight input variables, *exps_f12* [7] was considered. Its reduced two-level equation is given as,

$$exps\_f12 = a'b'c'd'efg'h' + a'bcd'e'f'gh' + a'c'defgh + a'b'c'de'f'gh + a'cd'e'f'g'h + a'bcd'f'g'h \quad (6)$$

TABLE VI
POWER DISSIPATION OF DIFFERENT IMPLEMENTATIONS
(FOR TYPICAL CASE: SUPPLY = 1.2V, TEMPERATURE = 25°C)

Implementation style	Power dissipation components		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
imp1	21.6456	21.3671	278.431
imp2	14.6222	14.434	188.172
imp3	10.9636	10.8101	153.499

As is usual practice to estimate the speed performance of a

gate based on its fan-out, a similar approach was used while assigning the drive strengths for the inverters, to be associated with the primary circuit inputs. The technology-mapped realizations correspond to a sort of leaf-DAG logic structure here, where DAG-ness is exhibited only in the primary circuit inputs. A similar structural representation was used for all subsequent case studies.

TABLE VII
POWER DISSIPATION OF DIFFERENT IMPLEMENTATIONS
(FOR BEST CASE: SUPPLY = 1.32V, TEMPERATURE = 0°C)

Implementation style	Power dissipation components		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
imp1	28.0776	27.2295	848.132
imp2	18.9056	18.3342	571.447
imp3	14.1289	13.6645	464.402

TABLE VIII
POWER DISSIPATION OF DIFFERENT IMPLEMENTATIONS
(FOR WORST CASE: SUPPLY = 1.08V, TEMPERATURE = 125°C)

Implementation style	Power dissipation components		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
imp1	17.5306	16.8477	682.884
imp2	11.8997	11.4401	459.591
imp3	8.96343	8.58947	373.961

The power consumption components for the three different mapped structures, corresponding to three different library cases (typical case, best case and worst case) for a 130nm UMC CMOS library are indicated in Tables 6, 7 and 8 respectively. The results mentioned in these tabular columns correlate quite well and add value to the above reasoning that a maximum fan-in based technology binding mechanism could potentially reduce power dissipation. However, this may be at the expense of an increase in delay; provided timing closure does not become a serious issue to reckon with.

VI. COMPARISON WITH OTHER FACTORING METHODS

Many of the cases referred to in this section are based on the examples cited in [9] [10] [11] [15] [21] [22]. Let us consider an example to illustrate the significance of the set theory based factoring scheme over a generic factoring scheme [11] in obtaining a multilevel solution.

Let Z be a Boolean function whose support set is dependent on four inputs and is given by,

$$Z = abc' + a'bc + abd + a'c'd + bcd \quad (7)$$

The different factored equations that could be obtained for Z based on an arbitrary choice of literals in succession are given by the following equations. The sequence of literals chosen has been mentioned alongside Z in parenthesis for each expression.

$$1^{st}\ sample - Z(a',b,d) = a'(c'd + bc) + b(d(a + c) + ac') \quad (8)$$

$$2^{nd}\ sample - Z(c,d,b) = c(bd + a'b) + d(ab + a'c') + abc' \quad (9)$$

$$3^{rd}\ sample - Z(d,b) = d(a'c' + b(a + c)) + b(a'c + ac') \quad (10)$$

The set theory based factorization method yields the following factored expression after three iterations of the above-mentioned algorithm.

$$Z = b(a(c' + d) + c(a' + d)) + a'c'd \quad (11)$$

The different power dissipation components of the above realizations (implemented using the high-density standard cells of a 130nm UMC CMOS process) for a typical case with a frequency of 100MHz and a supply voltage of 1.2V with automatic wire load selection is listed below.

TABLE IX
POWER CONSUMPTION OF DIFFERENT FACTORED EXPRESSIONS

Factored form	Power consumption parameters		
	Total Power (μW)	Dynamic Power (μW)	Leakage power (nW)
1st sample	6.38032	6.32669	53.6337
2nd sample	6.25973	6.20598	53.743
3rd sample	6.66331	6.60807	55.2453
Proposed	5.50443	5.4587	45.7286

Firstly, it is worth mentioning that the generic factoring scheme could enable a similar power optimal realization as that of the proposed one for the literal sequence *(a,b,c)*. However, it is clear from the above, that a literal factorization scheme suffers from the disadvantage that it could lead to many different solutions based on the choice of order of literals, though it is considered to be a faster scheme. As a result, the power quality of the realization may not necessarily be optimal for a random choice and so the selection of an appropriate literal sequence from a power perspective, for this method, would in turn introduce complexity as it could not be easily predicted at the technology-independent stage. On the other hand, the proposed set theory based factoring scheme leads to a single parenthesized expression in most cases and might be economical in terms of power dissipation. This is substantiated by the values listed in Table 9. There is a savings in total power by 12.06%, dynamic power by 12.04% and leakage power by 14.91% for the proposed method over the best of other realizations listed in the above table, corresponding to random ordering based on the generic factoring method.

Now, we take a function to examine the power quality of the factored forms obtained by X-factor (XF) [14] [15], Quick-factor (QF) [14] [15] and proposed methods.

$$Y = ac + ad + bc + bd + ce + cf + ae + ag + be + df + dg + bf \quad (12)$$

The factored expressions generated by XF, QF and

proposed algorithms are given by (13), (14) and (15) respectively. Their corresponding power values (obtained under a similar simulation environment) are listed in Table 10.

$$Y_{XF} = (c + d + e)(a + b) + (b + c + d)f + (a + d)g + ce \quad (13)$$

$$Y_{QF} = g(a + d) + (c + d + e)(a + b) + c(f + e) + f(b + d) \quad (14)$$

$$Y_{Proposed} = b(c + d + e + f) + a(c + d + e + g) + c(f + e) + d(g + f) \quad (15)$$

TABLE X
POWER CONSUMPTION OF DIFFERENT FACTORED SOLUTIONS

Factored form	Power consumption parameters		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
XF	4.48729	4.44613	41.1619
QF	5.21768	5.17031	47.3676
Proposed	4.30493	4.26404	40.8979

The proposed factoring procedure took eight iterations to obtain (15). From the above table, it can be seen that it results in savings in power consumption in comparison with the other schemes. For this particular example, the literal factoring scheme would have been able to obtain a similar solution as that of (15) for any ordering of single literal divisors.

Let us consider another example to highlight the usefulness of the proposed heuristic by considering a function X with a support set of nine variables.

$$X = abfg + aceg' + abeg' + abe'g + ace'g + acfg + dfg + deg' + de'g + bi + ch + ci + bh \quad (16)$$

$$X_{XF} = (a(b + c) + d)(eg' + g(f + e')) + (b + c)(h + i) \quad (17)$$

$$X_{QF} = (a(g(e' + f) + eg') + i + h)(b + c) + d(g(e' + f) + eg') \quad (18)$$

$$X_{LF} = a(b(eg' + g(e' + f)) + c(eg' + g(e' + f)) + b(h + i) + c(h + i) + d(eg' + g(f+e')) \quad (19)$$

$$X_{Proposed} = (g'e + g(e' + f))(a(b + c) + d) + (b + c)(h + i) \quad (20)$$

The factorized expression (19) was obtained for a random literal ordering *(a,d,b,c,g)*. The power consumption (under a similar simulation environment) of the above equations is listed in Table 11.

TABLE XI
POWER CONSUMPTION OF VARIOUS FACTORED FORMS

Factored form	Power consumption parameters		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
XF	7.53665	7.47397	62.687
QF	9.07217	8.99763	74.542
LF	14.4695	14.3473	122.233
Proposed	7.50666	7.44387	62.79

In this case, (20) was obtained in the eleventh iteration of the proposed heuristic. The QF heuristic is quicker and more preferred than the XF algorithm. However, for this case, it can be observed from Table 11, that the power quality of the XF algorithm is comparable with that of the proposed one and enables reduction in leakage power alone by 0.16%; while in terms of total power and dynamic power, it exhibits a slight increase by 0.39% and 0.4% respectively.

We now compare the results of the realizations based on Good-factor (GF) [9] [15], QF and proposed technique for a Boolean function of seven variables, given as follows.

$$W = pr + sp + rq + qs + tq + pt + pv + qu + tr + ru + vs \quad (21)$$

The respective factored forms, obtained on the basis of the above mentioned algorithms are given by the following equations in order.

$$W_{GF} = (r + s + t)(p + q) + u(q + r + s) + v(p + s) + rt \quad (22)$$

$$W_{QF} = (p + s)v + (p + q)(s + t + r) + r(t + u) + u(q + s) \quad (23)$$

$$W_{Proposed} = q(r + s + t + u) + p(r + s + t + v) + r(u + t) + sv \quad (24)$$

The corresponding power components of the above realizations for a 0.13μm process under similar conditions are indicated in Table 12. From Table 12, we find that the GF heuristic has resulted in a power optimal solution in comparison with the QF algorithm. However, the set theory based factorization procedure enables savings in total, active and static power components by 17.29%, 17.33% and 13.58% in comparison with that of the GF algorithm.

TABLE XII
POWER CONSUMPTION OF THE DIFFERENT FACTORED FORMS

Factored form	Power consumption parameters		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
GF	4.47287	4.43163	41.2378
QF	5.21768	5.17031	47.3676
Proposed	3.69913	3.66349	35.6357

Based on experiments on a large set of functions [15], it has been observed that GF algorithm is almost three times slower than QF due to the additional run-time spent in computing all of the kernels of the function before choosing a divisor. Though Boolean factoring enables good results, it is almost four times slower than GF algorithm owing to the extra time spent performing a strong Boolean division as opposed to a weak algebraic division.

Though the lesser-known factorization heuristic, FACT of [22] is found to be faster than the QF-algorithm based on experiments with arbitrarily generated functions, it deals with a different procedure for kernel-extraction based on the common rectangle-covering and intersection schemes. Heuristics are used in the three procedures of which this method is made up

in order to obtain a good factorized solution. The method is based on the generation of some products covering a set of true cubes suitably chosen. This choice may pose time complexity for large functions. The products forming a near-optimal factored expression are locally chosen from the ones covering each of those cubes.

Let us consider a sizeable logic function description, *R* of 12 inputs, given by (25) as,

$$R = y2y3y4y7 + y2y3y11 + y3y5y6y7 + y3y5y6y11 + y3y6y9 + y3y4y7y12 + y2y5y6y8y12 + y2y10 + y4y6 + y4y8 + y4y10 + y5y6y8y10 + y6y10y11 + y8y12 + y9y12 + y1 + y7y11y12 + y7y8 \quad (25)$$

$$R_{QF} = y3(y2(y4y7 + y11) + y6(y5(y7 + y11) + y9) + y4y7y12) + y2(y5y6y8y12 + y10) + y4(y8 + y6 + y10) + y1 + y10(y6(y5y8 + y11) + y12(y8 + y9) + y7y11) + y7y8 \quad (26)$$

$$R_{FACT} = x1 + x8(x4 + x7 + x2x5x6x12) + x10(x2 + x4 + x6x4 + x6(x11 + x5x8) + x11x7 + x12(x9 + x8)) + x3(x2(x4x7 + x11) + x6(x9 + x5(x7 + x11)) + x7x4x12) \quad (27)$$

$$R_{Proposed} = y6(y3(y5(y7 + y11) + y9) + y5y8(y2y12 + y10)) + y4(y3y7(y2 + y12) + y6 + y8 + y10) + y1 + y7y8 + y11(y2y3 + y6y10 + y7y12) + y2y10 \quad (28)$$

TABLE XIII
POWER DISSIPATION OF VARIOUS FACTORIZED EXPRESSIONS

Factored form	Power consumption parameters		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
QF	14.4082	14.2876	120.534
FACT	12.9385	12.821	117.562
Proposed	11.7189	11.6162	102.763

The factorized expressions obtained on the basis of the proposed technique betters the solutions rendered by other algebraic factoring methods in terms of power dissipation as can be seen from Table 13.

Finally, we consider an arbitrary function with a support set comprising 9 inputs, given by (29). Here, we compare the solutions obtained by the multi-level logic synthesis method outlined in [21] with that of our proposed method. The respective Boolean equations are given below.

$$S = abce + abde + ace'i + ade'i + a'fgh \quad (29)$$

TABLE XIV
POWER DISSIPATION OF THE FACTORIZED EXPRESSIONS

Factored form	Power consumption parameters		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
Approach of [21]	4.94834	4.9024	45.942
Proposed	4.50532	4.46411	41.2071

$$S_{Conventional} = a'fgh + (ac + ad)(be + e'i) \quad (30)$$

$$S_{Proposed} = a((be + e'i)(c + d)) + a'fgh \quad (31)$$

A wide variety of non-regenerative Boolean functions mentioned in previous literatures were considered in order to evaluate the significance of the proposed factoring scheme based on the set theory paradigm. The power consumption results obtained underline its usefulness.

VII. POWER ESTIMATION METHODOLOGY AND POWER DISSIPATION RESULTS OF COMBINATIONAL BENCHMARKS

Minimized two-level logic expressions for many MCNC/IWLS combinational benchmark functions [7] [8] and sub-functions were first obtained using Espresso tool [1]. Since the focus here is exclusively on analyzing the effect of factorization, the different functions/sub-functions were reduced depending on whether the normal output phase or the complementary output phase resulted in less number of essential prime implicants for each individual function output. This subsequently translates into less number of instances (library cells) for each function/sub-function output. The minimized equations resulting from Espresso are then given as the input expressions for the set-theory based factorizer tool, implemented using Java, and can be run on any platform. The outputs are the factorized Boolean equations, obtained after compilation. After a simple technology mapping of the non-factored and factored expressions, based on the methodology (maximum fan-in dependent library cell binding) discussed in section 5, power analysis was carried out using Synopsys PrimePower with a 130nm UMC CMOS technology library as the target for the typical case at an ambient temperature of 25°C; the recommended supply voltage for the typical case being 1.2V. The input clock frequency was set to 100MHz and the wire loads were selected automatically by the tool. The power dissipation results (total, dynamic and leakage) obtained for the benchmark functions are mentioned in Table 15, while those corresponding to the sub-functions are highlighted in Table 16 (both given in appendices).

Table 15 basically reports mixed results in terms of power consumption. This is because, in many cases, factorization tends to increase the number of cells needed for custom implementation while compared to a simple maximum fan-in based mapping for original reduced two-level logic. However, as the description set of each cube grows in size and when the function tends to comprise more cubes, the balance is tilted in favour of factorization. But it is clear that a factored solution would tend to exhibit a lower critical path delay in comparison with a non-factored solution comprising probably maximum fan-in cells in its longest path. Overall, we find that there is some appreciable reduction in leakage power by 13.48% over a pure technology mapped implementation; mean savings in total power and dynamic power being 2.91% and 2.79%. An important observation to be recorded here is that logic functionalities with embedded EXOR logic tend to benefit the

most from factorization operation. This is strongly justified by the power results of benchmarks *newtpla, newtpla2* and to some extent in the case of *newtpla1*. This is also visible in the case of sub-functions (refer Table 16) such as *5xp1_f1, br1_f3, br2_f8* and *sao2_f1*, which are EXOR intensive. It has been inferred that factorization does reduce leakage power considerably in cases where there is good sharing of literals among the essential prime implicants. This is evident from the results corresponding to *newtpla1* and *newtag* apart from *newtpla* and *newtpla2*. This is obviously good news for technology nodes pertaining to 90nm and below, where leakage power forms a significant proportion of the total power dissipation. Surprisingly, there are cases where leakage power is alone reduced even though an increase in dynamic and total power consumption are noticed for e.g. *newill* and *dekoder*, while the contrary is observed for *dc1*.

On an overall basis, the simulation results detailed in Table 16 report mean savings in total power, dynamic power and static power by 9.84%, 9.41% and 33.15% respectively. Considering sub-functions (individual outputs of benchmarks), significant reduction in all the three power components is noticeable for many functions, which have more non-redundant cubes and also exhibit good sharing of literals. To identify some, this is evident from the results corresponding to *br1_f3, br2_f8, m1_f9, m2_f11, root_f2, sao2_f1 and x1dn_f5*. There are also cases where reduction in static power alone can be observed for some sub-functions after factorization, even though an increase in total power and dynamic power is found. Samples for this include *amd_f22, newcpla_f4*, *dc2_f2*, *soar_f82, newcpla2_f3, misex1_f6* and *vtx1_f4*.

VIII. Summary and Conclusion

This paper highlights the importance of factorization as a basic and important step for multilevel logic realization of a combinatorial function and also various existing algebraic factoring techniques. The proposed factoring technique based on a set theory paradigm has been presented. The advantage of the proposed technique in obtaining an efficient factored solution for a given function and its power optimality was also demonstrated through experimental results. It is to be noted that though this method generates a unique solution, the possibility of obtaining multiple solutions is not ruled out. However, such expressions would all be not only logically equivalent but algebraically as well. The base-function set considered for simple technology mapping of a traditional logic realization, in accordance with the actual individual gate primitives available in a physical standard cell library is then discussed. This is followed by an analysis of the effect of factorization for many benchmark functionality implementations, for a typical case library specification.

The simulation results obtained for combinational benchmark functions and sub-functions indicate moderate average savings in total and dynamic power consumption parameters by 6.11% and 5.85% for the factored solutions over the non-factored ones. In terms of the leakage power component, considerable mean savings of 23.48% was reported. The novel factoring technique presented in this paper is expected to be a precursor for further research in combinational logic optimization, especially from a low power perspective.

References

[1] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, *Logic minimization algorithms for VLSI synthesis*, Kluwer Academic publishers, Boston, 1984.

[2] Sasan Iman, and Massoud Pedram, *Logic synthesis for low power VLSI designs*, New York, Springer Publishing, 1998.

[3] Farzad Nekoogar, and Faranak Nekoogar, *From ASICs to SOCs: A Practical Approach*, Prentice-Hall, 2003.

[4] T. Kuroda, "Low power high speed CMOS VLSI design," *Proc. IEEE International Conf. on Computer Design*, pp. 310-315, 2002.

[5] Semiconductor Industry Association's International Technology Roadmap on Semiconductors, Available: http://www.sia-online.org/backgrounders_itrs.cfm

[6] A.P. Chandrakasan, S. Sheng, and R.W. Broderson, "Low power CMOS digital design," *IEEE Journal of Solid State Circuits*, vol. 27(4), pp. 473-484, April 1992.

[7] K. McElvain, "IWLS '93 Benchmark Set: Version 4.0," distributed as part of the *MCNC International Workshop on Logic Synthesis*, Benchmark distribution, May 1993.

[8] S. Yang, "Logic synthesis and optimization benchmarks User Guide version 3.0," *MCNC Research Triangle Park*, NC, January 1991.

[9] R.K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proceedings of the IEEE*, vol. 78(2), pp. 264-300, February 1990.

[10] R.K. Brayton, and C. McMullen, "The decomposition and factorization of Boolean expressions," *Proc. IEEE International Symposium on Circuits and Systems*, pp. 49-54, 1982.

[11] R.K. Brayton, "Factoring logic functions," *IBM Journal of Research and Development*, vol. 31(2), pp. 187-198, March 1987.

[12] T. Stanion, and C. Sechen, "Boolean division and factorization using binary decision diagrams," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 13(9), pp. 1179-1184, September 1994.

[13] S.C. Chang, and D.I. Cheng, "Efficient Boolean division and substitution using redundancy addition and removing," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 18(8), pp. 1096-1106, August 1999.

[14] E.M. Sentovich, K.J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P.R. Stephan, R.K. Brayton, and A.L. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," *Electronics Research Laboratory Memorandum No. UCB/ERL/ M92/41*, University of California, Berkeley, May 1992.

[15] R.K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "MIS: A Multiple-level logic optimization system," *IEEE Trans. on CAD*, vol. 6(6), pp. 1062-1081, November 1987.

[16] K. Keutzer, "DAGON: Technology binding and local optimization by DAG matching," *Proc. 24th ACM/IEEE Design Automation Conference*, pp. 341-347, 1987.

[17] G. Merrett, and B. Al-Hashimi, "Leakage power analysis and comparison of deep submicron logic gates," *Proc. PATMOS, Lecture Notes in Computer Science*, Springer, vol. 3254, pp. 198-207, 2004.

[18] T. Sakurai, and A.R. Newton, "A simple MOSFET model for circuit analysis," *IEEE Trans. on Electron Devices*, vol. 38(4), pp.887-893, April 1991.

[19] B. Sheu, D.L. Scharfetter, P.K. Ko, and M.C. Jeng, "BSIM: Berkeley Short-Channel IGFET Model for MOS transistors," *IEEE Journal of Solid State Circuits*, vol. 22(4), pp. 558-566, Aug. 1987.

[20] R.X. Gu, and M.I. Elmasry, "Power dissipation analysis and optimization of deep submicron CMOS digital circuits," *IEEE Journal of Solid State Circuits*, vol. 31(5), pp. 707-713, May 1996.

[21] R. Rudell, "Logic synthesis for VLSI design," *PhD thesis*, University of California, Berkeley, 1989.

[22] G. Caruso, "Near optimal factorization of Boolean functions," *IEEE Trans. on CAD*, vol. 10(8), pp. 1072-1078, August 1991.

Padmanabhan Balasubramanian completed his B.E degree in Electronics and Communication Engineering from University of Madras, TN, India in 1998 and his M.Tech in VLSI System from National Institute of Technology, Tiruchirappalli, TN, India in 2005. He was earlier Lecturer in the School of Electrical Sciences at Vellore Institute of Technology (University and IET, UK Accredited), Vellore, TN, India. He is pursuing his research in the School of Computer Science at The University of Manchester, UK. His research interests are in combinational logic synthesis and optimization for low power and/or high-performance, and CMOS based digital IC design. He is a student member of IET, UK and IEEE, USA.

Ryuta Arisaka completed his B.Sc in Computer Science from The University of Manchester, UK in 2006. Subsequently, he received scholarship from the School of Computer Science of the same University to pursue his M.Phil (by Research) in the area of exact real arithmetic. His areas of interest include software programming, digital logic and numerical computation.

APPENDIX I

TABLE XV
POWER DISSIPATION RESULTS FOR COMBINATIONAL BENCHMARK FUNCTIONS

Benchmark function and specification	Technology mapped solution before algebraic factoring			Technology mapped solution after proposed algebraic factoring		
	Total power (μW)	Dynamic power (μW)	Leakage power (nW)	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
newtpla1 (10 inputs, 2 outputs)	7.08531	6.98426	101.05	5.14433	5.09453	49.8013
con1 (7 inputs, 2 outputs)	6.00884	5.94826	60.5824	10.1577	10.0707	86.9986
clpl (11 inputs, 5 outputs)	10.1231	10.0131	109.955	17.1432	16.9949	148.267
newtpla2 (10 inputs, 4 outputs)	31.3051	30.9902	314.945	15.766	15.607	158.963
newcwp (4 inputs, 5 outputs)	14.0829	13.9341	148.795	18.4418	18.2683	173.528
newtag (8 inputs, 1 output)	4.02158	3.96504	56.5371	2.40032	2.37696	23.3592
dc1 (4 inputs, 7 outputs)	38.5554	38.2694	285.999	38.4361	38.1092	326.887
newill (8 inputs, 1 output)	9.23922	9.10984	129.388	10.986	10.8783	107.648
newtpla (15 inputs, 5 outputs)	44.7396	44.2494	490.222	28.7395	28.444	295.509
squar5 (5 inputs, 8 outputs)	25.9126	25.6296	282.992	36.5106	36.1803	330.388
dekoder (4 inputs, 7 outputs)	18.4578	18.2578	200.02	19.7105	19.5252	185.273
Total	209.53145	207.351	2180.4855	203.43605	201.54939	1886.6221
Average	19.04831	18.85009	198.22595	18.49419	18.32267	171.5111

APPENDIX II

TABLE XVI
POWER DISSIPATION RESULTS FOR COMBINATIONAL BENCHMARK SUB-FUNCTIONS

Combinational benchmark sub-function	Number of primary circuit inputs	Technology mapped solution before algebraic factoring			Technology mapped solution after proposed algebraic factoring		
		Total power (μW)	Dynamic power (μW)	Leakage power (nW)	Total power (μW)	Dynamic power (μW)	Leakage power (nW)
5xp1_f1	7	7.36278	7.27808	84.6932	5.86574	5.81771	48.0295
amd_f22	14	11.4133	11.2622	151.164	11.7963	11.694	102.242
br1_f3	12	12.1521	11.9972	154.945	8.41237	8.31804	94.3269
br2_f8	12	7.49298	7.39024	102.735	3.94563	3.907	38.6291
bw_f26	5	5.39977	5.34586	53.9115	7.00439	6.94513	59.2615
m1_f9	6	4.00619	3.84971	156.483	2.75895	2.69453	64.4213
dc2_f2	8	9.58236	9.46676	115.6	10.0709	9.97475	96.1512
dk27_f3	9	3.54558	3.50826	37.312	3.83562	3.80283	32.7991
f51m_f5	8	4.98813	4.94554	42.5922	7.02084	6.96297	57.8722
inc_f2	7	6.48578	6.40998	75.7922	10.6611	10.5664	94.6656
luc_f23	8	6.79952	6.71101	88.5138	6.54115	6.48454	56.6071
m2_f11	8	5.66495	5.59397	70.9773	2.31539	2.2946	20.7905
misex1_f6	8	6.88917	6.80685	82.3142	6.98665	6.92824	58.4088
newcpla1_f4	9	3.29517	3.2459	49.269	3.35201	3.32252	29.4896
newcpla2_f3	7	7.27973	7.19907	80.6594	7.86389	7.79756	66.3265
newxcpla1_f7	9	3.53042	3.48815	42.2714	6.36686	6.31284	54.0174
opa_f60	17	5.36569	5.30566	60.0258	4.52687	4.48789	38.9804
rd53_f1	5	2.27182	2.24103	30.7932	3.92851	3.8898	38.71
risc_f1	8	4.7356	4.41938	54.1792	3.5534	3.51627	37.1295
root_f2	8	8.6056	8.51723	88.3768	5.8709	5.82283	48.068
sao2_f1	10	23.7311	23.46	271.075	15.0063	14.8571	149.225
soar_f82	83	3.25533	3.20841	46.914	3.73103	3.6972	33.8289
vtx1_f4	27	4.37461	4.30185	72.761	5.87272	5.81967	53.0492
wim_f2	4	2.8918	2.86076	31.037	2.41096	2.38795	23.0105
newtpla_f1	15	5.83013	5.76116	68.9673	5.18956	5.13955	50.0066
x1dn_f5	27	13.027	12.8837	143.374	7.37297	7.31047	62.4921
Total	341	179.97661	177.45796	2256.7365	162.26101	160.75239	1508.5385
Average	≈13	6.92218	6.82531	86.79756	6.24081	6.18278	58.02071