Energy Characterization based on Clustering

Huzefa Mehta, Robert Michael Owens, Mary Jane Irwin
Department of Computer Science and Engineering,
The Pennsylvania State University, PA 16802

Abstract

We illustrate a new method to characterize the energy dissipation of circuits by collapsing closely related input transition vectors and energy patterns into capacitive coefficients. Energy characterization needs to be done only once for each module (ALU, multiplier etc.,) in order to build a library of these capacitive coefficients. A direct high-level energy simulator or profiler can then use the library of pre-characterized modules and a sequence of input vectors to compute the total energy dissipation. A heuristic algorithm which performs energy clustering under objective constraints has been devised. The worst case running time of this algorithm is $O(m^3n)$, where m is the number of simulation points and n is the number of inputs of the circuit. The designer can experiment with the criterion function by setting the appropriate relative error norms to control the 'goodness' of the clustering algorithm and the sampling error and confidence level to maintain the sufficiency of representation of each cluster. Experiments on circuits show a significant reduction of the energy table size under a specified criterion function, cluster sampling error and confidence level.

1 Introduction

With the advent of portable computing, battery energy has become a key commodity. Circuits and systems now focus on power sensible designs. In order, to assist the designer to meet the power budget, it is necessary to have accurate and efficient power estimation tools. Power estimation is difficult because it is strongly data dependent. Existing tools [1, 4, 5, 6, 7, 10] operating at architectural and system levels, perform power estimation from high-level abstracted models of modules by characterizing the estimated capacitance that would switch when a given module is activated. [10] perform energy analysis of programs by direct simulation using instruction level power models. [8] describes a simulation approach where 'average' power costs are assigned to individual modules, in isolation from other modules. The power costs in this approach can add up to significant error and this model also ignores correlation between different modules. [1] uses a similar power profiling/simulation approach (PPROF), but uses small black box models of each module and does a lookup and evaluate for each module using the transition vectors computing energy per transition rather than average energy. [7] uses Uniform White Noise (UWN) in the technique called Power Factor Approximation to derive the switching capacitance for the modules based on random inputs and fit the capacitive coefficients to the results. This technique also does not account for strong dependency of power consumption on the input statistics. [4, 5] describes the Dual Bit Type (DBT) model in the technique called Stochastic Power Analyzer (SPA) which accounts for the random activity of the least significant bits and also the correlated activity of the most significant bits which also include the sign bit and hence has several capacitive coefficients. Although accurate, the problem with this approach is that it is not always possible to know the input statistics beforehand. Though, this method may work in DSP applications where signal statistics are predictable, it may not work in control driven environments such as in a general purpose microprocessor. Their subsequent work [6] describes Activity-Based Control (ABC) which considers activities of each module is obtained from functional simulation in this case and then the model is updated. The models of each module in [6] is built by taking into account the functionality whereas our characterization does not require any knowledge of the module functionality. The errors of characterization can be set by the designer and is based on clustering of input transitions.

Input statistics are highly dependent on the context of the usage of the circuit and the application the circuit is run on. So, high-level models of modules which are derived from input statistics may not match that of the environment they operate in. Hence, despite having accurate power models for individual modules a great deal of *overall accuracy* may be sacrificed, a weakness arising from unavailable or incorrect data statistics or inability to model the module interactions correctly. In case of a microprocessor driven by instructions it is impossible to know apriori the typical input patterns to the individual modules. Our characterization approach is based on the paradigm of direct simulation and profiling and hence is based on input transitions rather than input

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

statistics, without using any knowledge of module functionality and with the accuracy controlled by the designer. This is what distinguishes our approach from earlier works. This pre-characterization work is needed only once for each module without prior knowledge of any input statistics. The final outcome of the energy characterization work would be to establish a tool which takes as input stimuli, instructions to a processor and then simulates/profiles the energy consumption of individual instructions executing on that processor by looking up the appropriate energy coefficients (clusters) corresponding to the transition vectors. Instruction energy trace and statistics such as average energy per instruction or module, provided by this tool would be valuable information for architects to design low power processors compiler writers to retarget code for low-power [10] and application developers to compare the power consumption of various algorithms and data structures. However, the first step in doing such a energy simulation involves accurate and efficient energy characterization of modules based on input transitions. This paper concentrates on establishing such a method which groups closely related energy patterns to obtain clusters which to characterize the circuit. A comprehensive solution is presented, which includes efficient algorithms to solve the clustering problem and the sub-problems associated with it, while maintaining the quality and 'adequacy of representation' of the clusters.

Section 2 describes the clustering approach to module energy characterization. In section 3 the implementation is discussed and the results of the algorithms are presented. Finally we end with conclusions in section 4.

2 Module Energy Characterization

For the energy simulation/profiling methodology to be successful it is necessary to estimate accurately and quickly the energy dissipation of a module per input transition. The energy dissipation of a combinational circuit depends on the previous and present input vectors while that of a sequential circuit also depends upon the previous state vector. A circuit with n inputs and s storage elements (flip-flops) has 2^n possible previous and present input vectors and 2^s possible states. A fully characterized energy transition matrix of a combinational circuit hence has 2^{2n} entries whereas that of a sequential circuit has 2^{2n+s} entries. The clustering algorithms presented compress this energy matrix thereby reducing the number of entries needed to represent the circuit. Although, the final results are shown only for combinational circuits, this approach applies to sequential circuits also. When considering sequential circuits, only the manner in which the entries in the energy matrix are obtained (since the state vector will have to be taken into account) needs to be changed.

The problem with the simple approach of having a table lookup of a fully characterized energy transition matrix, is that the size of the characterizing energy table grows exponentially with the number of inputs of the circuit (and states of the circuit). It is not possible to store the energy values for this table, let alone simulate the circuit for all entries for large n. By observing these matrices, one notices that a lot of the entries have similar values for certain input transitions. By collapsing the 'bit patterns' of closely related energy transitions the number of values needed to store the matrix can be reduced. Furthermore, if error can be tolerated then a lot of the patterns could be clustered together. Depending on the this error tolerance the 'lower order effects' of the matrix is retained and the 'higher order effects' is discarded thereby, reducing the number of characterizing coefficients.

The values in the energy matrix (z) are energy per transition obtained experimentally by running a circuit simulator (SPICE) or they can represent the switched capacitance values as measured by IRSIM-CAP. Energy is proportional to switched capacitances and can be obtained by scaling by V_{dd}^2. IRSIM-CAP [5] is an enhanced version of IRSIM and has been calibrated to within $10-15\%$ of SPICE and is used to obtain the switching capacitances for our experiments. The switched capacitances along with the previous and present input vector are read from a simulation file. The diagonal entries of the energy transition matrix are all zero since energy dissipation is zero when the input vectors do not change. The fully characterized energy table (switched capacitances) for a 2:1 multiplexer is shown figure 1 reveal some of the energy patterns in 3-D. The x-axis represents the present vector, y-axis the previous vector and z-axis the energy values. The values are scaled between black (least energy dissipation) and white (maximum energy dissipation). In order to identify the energy patterns one has to keep in mind that these patterns are distributed over input vectors which are in boolean space.

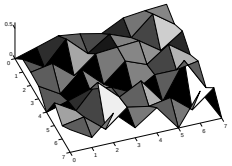

Figure 1: Energy matrix for 2:1 multiplexer

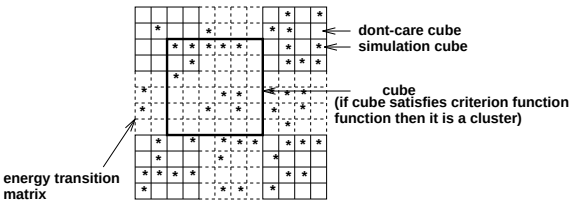

Figure 2: Definitions

2.1 Terms and definitions

Our commonly used terms and definitions (Figure 2) are given below.

- An *energy cube* or *cube* is a 3-tuple consisting of three vectors, viz., a set of previous input vectors, a set of next input vectors and the energy estimate of those transitions.
- A *simulation cube* is the cube corresponding to a simulation point. A simulation cube has only one previous and present input vector.
- When simulation is not performed for a particular input vector transition, the respective energy value in the matrix is undefined and is said to be a *dont-care cube*.
- A cube is said to be a *neighbor* of another cube if the number of bits of their input transition vectors differ by one.

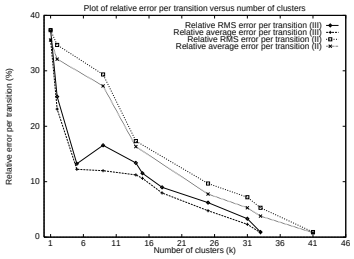

Figure 3: Relative error per cluster v/s number of clusters for 2:1 multiplexer

- A *criterion function* is some criteria which the energy values of the simulation cubes covered by a cluster satisfy. Some examples of criterion functions are given below
  - The energy per transition of each cube should obey certain relative error bounds (average, RMS, maximum)
  - The distance of energy values between cubes of a same cluster should be within a certain range
- The *energy estimate* of a cube is taken to be the mean of all the energy values the cube represents.
- A *cluster* is a cube such that the energy values of the simulation cubes it covers satisfies some criterion function.
- The *sampling error* of the cluster is the error bound allowed between the observed and actual energy estimate of the cluster.
- The *confidence level* of the cluster is the confidence that is placed in the energy estimate such that it is always bound by the sampling error of the cluster.

2.2 Criterion evaluation

In case of average energy or power, all transitions of the simulation cube distribution map to a single mean value $\bar{z} = \frac{\sum z}{m}$. The problem is to partition the complete boolean space of size 2^{2n} into a set of (say k) clusters $(C_1, \cdots, C_k)$ in a manner that sufficient peaks of clusters are retained with minimum impact in error while minimizing the number of clusters. The observed energy estimate $(\bar{z}_i)$ is the average mean of all the simulation cubes the cluster C_i represents and is defined as $\bar{z}_i = \frac{\sum_{z \in C_i} z}{n_i}$ where n_i is the number of simulation cubes the cluster represents. The designer can experiment with three error norms, average (ϵ_{r_1}), RMS (ϵ_{r_2}) and maximum (ϵ_{r_∞}) relative errors to control the quality of the clustering. The algorithm will accept bounds on any combination of the relative error per transition for each cluster. If a cluster satisfies all the applied error bounds, it is accepted. At the end of the clustering algorithm, the relative error per transition for the total distribution is output showing the total 'goodness' of the clustering. The plot of error per cluster versus number of clusters for 2:1 multiplexer is shown in figure 3. As expected the errors (both RMS and average) increase with the decrease in the number of clusters (for all the runs the maximum relative error was bound to 30%). However, note that for a better heuristic algorithm (III) the errors and the number of clusters are lesser than the inferior heuristic algorithm (II). Also for 0.5% error the number of clusters obtained by algorithm II is 41, while that obtained by algorithm III is 34. In the average case (i.e., one cluster), both the heuristics converge to the same error. The quality of the clustering can be decided by the designer by choosing the appropriate coefficients (c_1, c_2, c_3, c_4) of the following cost equation.

$$Cost = c_1 k + c_2 \epsilon_{r_1} + c_3 \epsilon_{r_2} + c_4 \epsilon_{r_\infty}$$

2.3 Coverage sufficiency

The clustering algorithm attempts to cover the entire boolean space. However, one problem is still unresolved. How do we say for sure that we have sufficient points representing each cluster? The cluster must contain enough simulation cubes to show that the observed energy estimate is a sufficient. Hence, in order to bound the actual $(\bar{z}_i)$ from the expected (μ_i) energy estimate by a sampling error of say α (i.e., $\frac{|\mu_i - \bar{z}_i|}{\mu_i} \leq \alpha$) with β% confidence level, a certain number of representative points are needed. The cluster C_i is represented sufficiently if $n_i \geq (\frac{c_\beta \sigma_i}{\bar{z}_i \alpha})^2$ [2, 3] where n_i is number of points represented by the cluster, σ_i is the standard deviation of the cluster and c_β is the standard normal variable for β% confidence interval ($c_{90\%} = 1.645, c_{95\%} = 1.96, c_{99\%} = 2.58$). The confidence level is set to 95% and the sampling error is set to 10% for our experiments.

2.4 Clustering algorithm

The clustering problem can be stated as follows: given the simulation cubes of a circuit, find the minimal number of clusters which satisfy a criterion function, coverage sufficiency requirement and cover the whole boolean space. Since it is not possible to store the entire matrix, all entries are internally represented as a list of simulation cubes. The circuit is simulated only for m random vectors where $m \ll 2^{2n}$. The clustering algorithm is then performed (Figure 4). If the cluster coverage is insufficient, the process is terminated. The circuit is simulated for additional m vectors and clustering is performed using the new points. The algorithm ends when the clusters satisfy the coverage sufficiency.

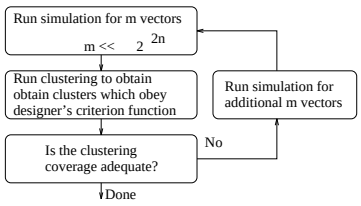

Figure 4: Overall methodology

Energy clustering is similar to a lot of different classical problems. The first is that of boolean logic minimization. In boolean minimization, cubes can be merged if they are neighbors and if they are ON (1) or dont-care (X). In energy clustering, cubes can be merged if they are neighbors and if the energy values of the merged cube satisfy some criterion function. A dont-care cube can merge with any neighboring cube and assumes the same energy value of the merged cube. A good criterion function tries to minimize the total error and the number of clusters. Clustering is similar to the test pattern generation problem by simulation based directed-searching. Clustering is also a well-defined problem in discrete optimization; find those partitions of the set of samples

that extremize the criterion function. Since the sample set is finite, there are only a finite number of possible partitions. Thus, in theory the clustering problem can always be solved by exhaustive enumeration. However in practice such an approach is unthinkable for all but the simplest problems. There are approximately $c^n/c!$ ways of partitioning a set of n elements into c subsets, and this exponential growth with n is overwhelming. For example, an exhaustive search for the best set of 5 clusters in 100 samples would require considering more than 10^{67} partitionings. Thus, in most applications an exhaustive search is completely infeasible. A lot of research has been done in clustering as a method of data analysis [3] which use various distance functions (Euclidean, Manhattan) and a variety of algorithms (nearest-neighbor, furthest-neighbor, minimum mean-squared-error) or a combination of them.

2.4.1 Exact algorithm

A straight forward way to form clusters is to merge locally and grow bottom-up. This is essentially similar to the Quine-McCluskey method for logic minimization [9]. Each cube is combined with its neighboring cube, if the combined cube satisfies the criterion function and cubes at all levels are combined till no further clusters can be formed. The algorithm starts at the lowest level cubes (simulation cubes). Neighbors are built for cubes at the current level. Merging takes place among neighboring cubes which satisfy the criterion function (*validCluster*). When cubes are merged, they are added to the next level. The merging process terminates if there is no merging at the current level. The final step is a greedy disjoint cover finding algorithm which starts at the maximum level and attempts to cover the whole matrix.

Building neighbors should take $O(m^2)$ running time, however this true, only if the merges take place over the simulation cubes. The objective however is to cover the whole boolean space therefore the merges also take place over all the 'empty' dont-care space. Hence, this operation takes $O(2^{2n^2}) = O(2^{4n})$ running time. There are atmost $2n$ invocations to this procedure (since the length of the previous vector and the next vector are n bits each), the algorithm hence takes $O(2^{4n}n)$ time. The procedure *validCluster* (Figure 6) checks if each simulation cube covered within the cluster satisfies the criterion function. Our present implementation of *validCluster* uses simple thresholds for the different relative errors. Therefore, *validCluster* is linear in the number of simulation cubes and the number of inputs $O(mn)$. If we had a complicated criterion function such as minimizing the maximum distance between simulation points in a cluster then *validCluster* would take $O(m^2n)$ time. This exact algorithm, is expensive and is exponential in the number of inputs of the circuit. Since the exact algorithm is not feasible, a faster heuristic algorithm is devised which will run in polynomial time.

2.4.2 Heuristic algorithm (clusterII)

The exact method (since it is exponential in the number of inputs) is not feasible for number of inputs greater than four. This leads to an approximate method (clusterII) (Figure 6) which reduces the running time significantly. This algorithm is a variant on the ESPRESSO-II, III minimization procedure which uses the EXPAND heuristic. A random uncovered simulated cube is chosen as a starting point (Figure 5). This point is expanded to every other uncovered simulated cube. This expanded cube is checked if it is a 'valid cluster'. If it is a 'valid cluster' and does not intersect with other already found clusters (since the clusters are disjoint), then it is chosen as one of the possible clusters. The cluster which is finally chosen, is the one which covers the most simulation cubes. If no expanded cube can be validated as a cluster then the starting cube is expanded arbitrarily into a cluster in order to cover the dont-care space. The simulation cubes covered by the chosen cluster are deleted from the list. The algorithm repeats, till all the cubes from the list are covered. The quality of the algorithm is sensitive on the initial starting cube (which is random). A bad initial cube might prevent the search for good cluster.

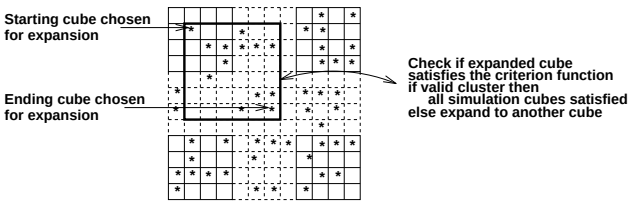

Figure 5: clusterII heuristic

The algorithm takes $O(m^3n)$ worst case time since expansion to each cube can be in the worst case $O(m^2)$ (when no merges can take place) and *validCluster* takes $O(mn)$ running time. This worst case running time occurs in the case when ALL the simulation cubes are clusters and this is highly unlikely. The expected running time is far less than the worst case running time as shown by the experiments.

2.4.3 Additional heuristics (clusterIII)

By observing a set of clusters it was observed that only a few points within a cluster violate the criterion function. If these points are included in the cluster then the accuracy per transition of the cluster suffers. If these points are to be excluded then the cluster cannot be formed resulting in the formation of many small clusters. The previous heuristic algorithm (clusterII) allows only disjoint clusters. This makes the algorithm strongly dependent on the initial starting point. Allowing overlapping clusters allows subsequent clusters to be bigger in size and cover more simulation cubes. An improvement is also obtained by eliminating the diagonal of the matrix (which is always zero) by marking them as done. By starting at the violating exceptions and allowing overlapping clusters, the number of clusters and the error per transition and overall error can be significantly reduced. The extension to clusterII (clusterIII) (Figure 6) to incorporate for overlapping clusters is very easy. The *notIntersect* check used in clusterII to maintain disjointness is not needed and already covered simulation cubes are not accounted for in *validCluster*. Subsequent cluster formations can then allow overlaps. In order to cover the exceptions first they have to be identified first. The question then arises is that 'How do we find these problematic exceptons?'. The clusterIII algorithm is modified in the following manner. First, all the m^2 possible expansions are enumerated. Then a precedance tree is established such that expanded cube A points to expanded cube B iff B is completely covered within A (Figure 7). The covering algorithm starts at the top of the tree (which is the complete boolean space). The current tree node (expanded cube) is checked if it is a 'valid cluster'. If it is a 'valid cluster', then the current node and all of the descendants are marked as covered and the next uncovered tree node is attempted for cover. If the node is not a 'valid

```
procedure validCluster (c)
begin
  if (numCovered() < reqNumCovered()) then exit(-1);
  if ((relErrs() < userErrors) then return (FALSE);
  return (TRUE);
end
algorithm clusterII
begin
  cubeList := readCubes();
  expandCubeList(cubeList);
end
procedure expandCubeList(cubeList)
begin
  clusters := 0;
  while cubeList do begin
    c := arbitaryCube(cubeList);
    cluster := expandCube(c, cubeList);
    foreach (d ∈ cubeList) do
      if (coveredBy(d,cluster))then
        cubeList := deleteFromList(d, cubeList);
    clusters := addToList(cluster, clusters);
  end
end
procedure expandCube(c, cubeList)
begin
  while (notCluster(c)) do begin
    expansions := 0;
    foreach (d ∈ cubeList) do begin
      exp := smallestCubeContaining(c, d);
      if (validCluster(exp)
      and notIntersect(exp,clusters)) then
        expansions := addToList(exp, expansions);
    end
    if (expansions=0) c := expandCubeArbitrarily(c);
    else c := selectMember(expansions, cubeList);
  end
  return (c);
end
algorithm clusterIII
begin
  cubeList := readCubes();
  enumeratedCubes := enumerateCubes(cubeList);
  precedanceTree :=
  buildPrecedanceTree(enumeratedCubes);
  c := topOfTree(precedanceTree);
  while (notCovered(c)) do cover(c);
end
procedure cover(c)
begin
  if (validCluster(c)) then begin
    markSelfandDescendantsCovered(c);
    return (TRUE);
  end else begin
    foreach (child ∈ children(c)) do
      if (validCluster(child)) return (TRUE);
    orderChildren(c);
    foreach (child ∈ children(c)) do
      if (cover(child)) then return (TRUE);
  end
  return (FALSE);
end
```

Figure 6: validCluster, clusterII, clusterIII

cluster', then its children are tested for 'valid clusters'. If none of the children of the node are 'valid clusters' then the children are ordered based on the violating error of the parent and the children are covered based on decreasing violating error. This process is repeated till all the tree nodes are covered.

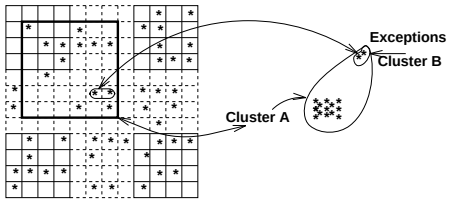

Figure 7: clusterIII heuristic

The running time of the algorithm, clusterIII is still $O(m^3n)$. Enumerating the cubes takes $O(m^2)$ time. Building the precedance tree does not take $O(m^3)$ time since the enumerated cubes are inserted in a bucket of their level and while building the tree only cubes of adjacent levels are checked against each other. The worst case scenario occurs when ALL the tree nodes call *validCluster* which takes $O(mn)$ time, hence the worst case running time of clusterIII is $O(m^3n)$.

3 Implementation and Results

The exact and the heuristic algorithms (clusterII, clusterIII) are implemented in C. The exact algorithm fails for inputs greater than four. The heuristic algorithms run in polynomial time $O(m^3n)$ where m is the number of simulation points and n is the number of inputs of the circuit. The input to the algorithm is a set of simulation cubes and the desired relative errors (average, RMS, maximum) the designer would like to control in the clustering algorithm. The cluster sampling error and confidence level is also specified. The output is the set of clusters and the relative errors for the individual clusters and the total distribution.

In table 1 we have results for 12 circuits. The second column shows the number of inputs (n), the third column shows the number of simulation points (m) required for sufficient representation of each cluster. The subsequent columns show the 'goodness' of algorithm clusterII and clusterIII in terms of the number of clusters (k), the CPU time in seconds and the relative errors for the total distribution obtained for the 'best' runs. The term 'best' to us meant trying to minimize the RMS error and the number of clusters while keeping an accepted maximum error bound per transition to 30%. As we see that the algorithm clusterIII which is similar to clusterII but uses the 'start at exception and allow overlap' heuristic gives better results than clusterII almost all the time. The acceptable confidence level is set to 95% for each cluster for 10% sampling error between the actual and observed energy estimate of the cluster. We see that while keeping the maximum error fixed with 30% and while playing the RMS error bound within 10%−15%, a significant reduction in number clusters is obtained. One should note that even one point with a large maximum error in ANY of the clusters, forces the cluster and the entire distribution to have that error although the RMS error can be relatively small. However, the maximum error is useful when a large spread cannot be tolerated. Another, important observation is that maximum error violations occur with clusters with small values. In all our circuits, the initial value of m was sufficient enough for cluster representation.

Table 1: Clustering results for best runs (n:no of inputs, m:simulation points, k:no of clusters)

Circuit	*n*	*m*	*clusterII*					*clusterIII*				
			k	*time* (secs)	*% relative error/transition* ave	RMS	max	*k*	*time* (secs)	*% relative error/transition* ave	RMS	max
mux21	3	64	14	1	13.33	14.33	28.34	5	2	12.24	13.24	29.93
fa	3	64	15	2	10.44	13.21	29.72	6	3	10.05	12.96	29.77
sda3	6	200	19	137	13.03	13.73	29.32	15	183	11.86	10.12	27.09
mult4	8	500	27	191	9.62	13.32	23.64	16	256	10.46	14.32	22.21
rca4	8	500	23	172	11.23	14.42	24.44	13	272	10.23	12.21	21.23
cla4	8	500	21	167	13.73	14.73	29.31	12	298	11.14	14.62	26.99
hpar	9	500	36	286	8.53	9.42	25.62	15	432	7.33	8.2	22.21
mux81	11	500	47	202	11.36	12.69	25.36	25	477	10.27	13.53	22.43
hcomp	11	500	49	272	12.67	12.98	27.93	23	456	11.13	13.4	21.61
halu	14	500	59	291	12.24	12.26	23.13	27	487	11.52	14.37	22.33
rca16	32	1000	173	4497	13.37	14.31	22.55	123	5239	12.32	13.62	29.54
cla16	32	1000	155	4263	12.39	14.20	21.35	97	5033	12.00	13.53	28.62

4 Conclusions

We have established a new method to cluster the energy table for modules. The discerning features of our work are that characterization is based on input transitions rather than input statistics, without knowledge of module functionality with the accuracy being controlled by the designer. Energy characterization needs to be done only once for each module. The heuristic algorithms devised achieve good compression of the energy tables while running in polynomial time. The input is a set of simulation cubes and criterion function established by setting bounds on either the average, RMS or the maximum relative error per input transition. The worst case time complexity of the algorithm is $O(m^3n)$ where m is the number of simulation cubes and n is the number of inputs of the circuit. The worst case scenario $O(m^3n)$ occurs when ALL the simulation cubes are clusters (highly unlikely). It is difficult to derive the average running time, however actual CPU time suggests that the average running time is much less than $O(m^3n)$.

As explained in section 2, although the results are shown only for combinational circuits, this table collapsing approach by clustering is also applicable to sequential circuits. The only thing which has to be modified is the method of obtaining the entries of the energy table (since for sequential circuits the energy is also dependent on the state vector).

Our next step is to build a library of such clustered tables for the modules. This library can then be used directly in in a simulation/profiling tool. We would like to work on the following as part of further research:

- The initial implementation of the algorithm is naive since the effort was concentrated on the establishing the methodology rather than obtaining an efficient algorithm. The running time of the algorithm can be reduced by incorporating better search mechanisms to detect exceptions and using better data structures.
- Using hints on the type of module functionality to decrease the number of clusters
- Work on obtaining a library of module clusters for datapath combinational modules
- Obtain results for benchmarks on sequential modules (register files, controller etc)
- Incorporate the module clusters into the simulation or profiling framework.

REFERENCES

[1] Huzefa Mehta, Robert Owens, and Mary Jane Irwin. Instruction level power profiling. In *International Conference on Acoustics, Speech and Signal Processing*, 1996. To appear.

[2] James T. McClave and Frank H. Dietrick. *A First Course in Statistics.* Macmillan Inc.,, 1989.

[3] Lothar Sachs. *Applied Statistics - A Handbook of Techniques.* Springer-Verlag, 1984.

[4] Paul E. Landman and Jan M. Rabaey. Power estimation for high level synthesis. In *EDAC-EUROASIC*, pages 361–366, 1993.

[5] Paul E. Landman and Jan M. Rabaey. Black-box capacitance models for architectural power analysis. In *Proceedings of the International Workshop on Low Power Design*, pages 165–170, April 1994.

[6] Paul E. Landman and Jan M. Rabaey. Activity-sensitive architectural power analysis for the control path. In *Proceedings of the International Workshop on Low Power Design*, pages 93–98, April 1995.

[7] Scott R. Powell and Paul M. Chau. Estimating power dissipation of VLSI signal processing chips: The PFA technique. In *VLSI Signal Processing IV*, pages 250–259, 1990.

[8] T. Sato, M. Nagamatsu, and H. Tago. Power and performance simulator: ESP and its applications for 100 MIPS/W class RISC design. In *Proceedings of the 1994 IEEE Symposium on Low Power Electronics*, pages 46–47, Oct 1994.

[9] Tsutomu Sasao. *Logic Synthesis and Optimization.* Kluwer Academic Publishers, 1993.

[10] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: First step towards software power minimization. In *Proc. of the Int'l Conference on Computer Aided Design*, pages 384–390, Nov 1994.