SA based Power Efficient FPGA LUT Mapping

Richard Dobson
King's College London
Department of Informatics
richard.dobson@kcl.ac.uk

Kathleen Steinhöfel
King's College London
Department of Informatics
kathleen.steinhofel@kcl.ac.uk

ABSTRACT

Look up Table (LUT) based Field Programmable Gate Arrays (FPGAs) are commonly used in mobile devices due to their efficient signal processing capabilities and flexibility to be reprogrammed in situ. However the mechanisms which enable a FPGA to be re-programmable make it require more power than an Application Specific Integrated Circuit. In this paper we consider the power reduction of a FPGA by optimising the mapping the underlying boolean circuit onto the LUT based FPGA with respect to cumulative switching. We formulate the power minimisation problem as a combinatorial optimisation problem. To tackle this NP hard problem we propose the application of a local search method. Here we introduce a complete a neighborhood function and apply heuristic simulated annealing in conjunction with the objective function from [20] 'cumulative switching'. Our experimental results show a 42.96% average reduction in power consumption compared to SIS based mapping and 27.44% average reduction in power consumption compared to a genetic algorithm.

Categories and Subject Descriptors

J.4 [**Computer Applications**]: Physical Sciences and Engineering—*Electronics*; B.7.1 [**Hardware**]: Integrated Circuits—*Types and Design Styles*

Keywords

Simulated Annealing, Evolutionary Algorithm, Power Efficient, FPGA, LUT

1. INTRODUCTION

Logic circuits are a integral part of computer science and are ubiquitous in everyday life. Many logic circuits are implemented using Application Specific Integrated Circuits which are developed for a specific purpose and once manufactured their function is fixed. FPGAs are logic circuits with additional technology which allows them to be programmed after manufacturing and reprogrammed (depending on the implementing technology); this is ideal for a multitude of applications including mobile and distributed computing.

As mobile or distributed devices usually rely on a limited power supply unit, such as a battery or renewable power source it has become increasingly important to develop FPGAs which are exceptionally power efficient. The power consumption of a FPGA can be broken down into static power and dynamic power. Static power is consumed whenever there is power running through the circuit regardless of activity. Dynamic power is consumed when the circuit is active and accounts for 62% of total power consumption for a Xilinx Spartan-3 device [25]. Table 1 outlines the power consumption of various components.

Dynamic Power	Static Power
Routing = 62%	Config = 44%
Clock = 19%	Routing = 36%
Logic = 10%	Logic = 20%
Other = 9%	

Table 1: Breakdown of power consumption of the Xilinx Spartan-3 device [25]

By far the largest proportion of the power consumption is the cost of dynamic routing which accounts for 38.44% of the total power consumption of a Xilinx Spartan-3 device. Power consumed by dynamic routing is dependent on the switching activity of the routing edges which is dictated by the switching activity of each logic block and the length of each path. In the majority of modern FPGAs logic blocks are implemented using Look Up Tables (LUTs) which take k_in inputs and return a single result.

There have been many algorithms developed which aim to reduce the amount of power consumed by dynamic routing. One of the most fruitful areas under consideration is the computationally hard problem of mapping an input boolean function (usually in the form of a boolean circuit) onto a LUT based FPGA such that the power consumption is minimised.

In this paper we develop a simulated annealing based algorithm which maps a boolean function onto a LUT based FPGA such that the power consumed through dynamic routing is minimised. The vast majority of existing solutions make use of greedy heuristic algorithms which have the disadvantage of often finding only locally optimal solutions rather than the global optimum. Simulated annealing based

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

algorithms have had significant success with other problems which have similar properties and has found globally optimal or near optimal solutions in an acceptable time period. We anticipated that it can provide equally strong results for our problem.

We begin the paper with an overview of relevant literature with a focus on algorithms which reduce the power consumed by dynamic routing. We then formally define the problem considered and briefly outline the power estimation technique used to evaluate solutions. Finally we introduce our Simulated Annealing algorithm which is tailored to the problem at hand. We compare our results to both SIS mapping [21] and a genetic algorithm [20] and show that our algorithm improves both of these results.

2. BACKGROUND

Boolean algebra was first proposed by George Boole in 1854 [3]. Boolean algebra was researched and developed over many years and was finally applied to digital circuits in 1937 by Shannon [22]. This was utilised in the development of FPGAs which were introduced in 1988 when Freeman filed a US patent on behalf of Xilinx Inc. [11] which describes a configurable logic circuit similar to modern day FPGAs.

Simulated annealing was presented as a operations research global optimisation method by Kirkpatrick et. al. in 1983 [13] (and later independently by Ĉerny [6] in 1985). It has since been applied to many combinatorial optimisation problems and has been responsible for improving the upper bound and finding optimal / near optimal results for many hard problems where other algorithms have failed. For example Steinhöfel, Albrecht and Wong [23] applied heuristic simulated annealing to the job shop scheduling problem; in the paper they showed that the algorithm could find optimal solutions for a number of benchmark problems and improved the best known solution for several more. For a good overview of Simulated Annealing algorithms, associated techniques and terminology see [1].

2.1 Low Power LUT based FPGA Mapping

By far the most vigorously researched area for reducing the power consumption of LUT based FPGAs has been to develop an algorithm which maps an input boolean function onto a LUT based FPGA. Farrahi and Sarrafzadeh [10] initiated the research by first showing that the decision version of the problem is NP complete even for simple classes of circuits (e.g. 3 level circuits), they then extended this to show that even restricted cases of the LUT minimization for FPGA technology mapping are NP-complete [9].

Farrahi and Sarrafzadeh [10] considered mapping boolean circuits onto LUT based FPGAs in 1994. The authors developed a heuristic algorithm (Power Min) which maps the nodes into k feasible cones (which are analogous to k feasible LUTs) whilst attempting to minimise average power consumption. It is shown that the heuristic described can reduce the power consumption by an average of 14.8% whilst using only 7.1% more LUTs compared to an algorithm designed to minimise area.

Wang and Kwan [26] suggested a heuristic mapping algorithm with the aim of reducing the power consumption whilst maintaining optimal area. The authors' algorithm first generates the LUT mapping which results in the least number of LUTs possible; the algorithm then adjusts the solution to hide the high transition paths inside LUTs which results in reduced power consumption whilst maintaining the number of LUTs. The algorithm reduces the power consumption by 10.38% and maintains the number of LUTs.

Wang, Liu, Lai and Wang [27] proposed Power-Map; a heuristic algorithm which relies on a limited cut enumeration technique to generate many potentially good solutions. The authors compare their Power-Map algorithm to the Power Min algorithm from [10]; Power-Map reduced the power consumption by 14.03% - 14.18% and the number of LUTs by 6.31% - 6.99% depending on then number of cuts.

Li, Mak and Katkoori [17] developed a multi objective technology mapping algorithm which aims to reduce the power consumption whilst ensuring that the circuit depth is kept optimally small. The authors exploit the fact that non critical LUTs can be altered without affecting the depth of the circuit. PowerMap is compared to a minimum depth mapping algorithm FlowMap; PowerMap reduces the power consumption by 17.8% and the number of LUTs by 9.4% with no depth penalty.

Anderson and Najm [2] developed a mapping algorithm which relies on cuts to decompose the boolean circuit. The authors also consider logic duplication which has been previously shown to be essential for minimum depth LUT circuits. The algorithm is compared to FlowMap (which mimimises depth) and FlowMap-r (which minimises depth whilst trying to reduce the total number of LUTs) the researchers find that their algorithm uses less power, area and connections.

Li, Mak and Katkoori [18] [16] develop a heuristic algorithm which attempts to minimise the power consumption of the mapping solution. The algorithm (Power Min Map / Power Min Map -d)) takes a global view when deciding which cut to accept at any point, opting for the cut which tends to reduce the power consumption of the overall circuit rather than just the best local cut. A network flow min cut method is used to compute the initial solution which is then adjusted to further reduce the power consumption. Power Min Map is compared to Power-Map [27]; the find that on average it reduces the energy by 12.2% and the number of LUts by 10.6%.

Pandey and Chattopadhyay [20] present the first stochastic algorithm to address the problem of FPGA LUT circuit mapping. The authors begin by reducing the problem to a binate covering problem and then use a genetic algorithm to search for a good solution.The authors compare their algorithm to a basic SIS (Berkley, Robert Brayton) map and find that they reduce the power consumption by 25.51%. The authors claim that the algorithm in [27] only reduces switching activity by 10% in comparison to the SIS solution.

Bucur, Stefanescu, Supateanu and Cupcea [5] design a mapping tool which builds on the SIS circuit tool. The authors approach differs from other in that it uses a Monte Carlo simulation to estimate the energy consumption rather than probability based approach which most publications use. The tool attempts to minimise the power consumption whilst also considering depth and area. The authors present 3 different solutions and compare them to one another, this makes it hard to compare this approach to those listed above.

Chen, Wei, Zhou and Cai [8] have developed a heuristic algorithm, PowerMap er which considers both power consumption and edge count simultaneously. The algorithm first generates all cuts for all nodes and maps a solution; it then applies an area-edge recovery method 'depth slack

distribution' and finally it recomputes the edge cost. The authors compare the algorithm to Power Min Map -d [16] (Power = -8.5%, Area = -8.4%) and MacroMap (an area optimal algorithm) [29] (Power = -18%, Area = -7%). Although we must bear in mind that the figures quoted in this paper are maximum improvement rather than average improvement as quoted in other papers.

2.2 Dynamic Routing Power Reduction

After an initial LUT mapping solution has been generated another layer of optimization can take place; the function of some LUTs can be subtlety altered such that the output switching is reduced but that the circuit as a whole still implements the same function. Chen, Hwang and Liu [7] were the first to look at this problem. They utilises Roth Karp decomposition along with local search techniques (Simulated Annealing and Kernigham-Lin) to modify the individual functions which each LUT implements whilst maintaining the same circuit function. The authors found that after applying their algorithms they achieved a greater than 9% average power reduction in comparison to the standard SIS mapping. There are several other papers which present solutions to the same problem [12] [14] [15] but for space considerations we do not discuss them here.

Mashayekhi, Jeddi and Amini [19] introduced methods which reduce switching within each LUT block by inserting fake registers and then using a re-timing method; the authors implemented their methods for two of ISCAS89 benchmark circuits and achieved a 25% power reduction over using similar re-timing methods without power reduction considerations.

Finally, Tinmaung, Howland and Tessier [24] developed logic synthesis methods to reduce power consumption; they achieved an average power reduction rate of 13% for Altura Cyclone II devices compared to the standard SIS logic synthesis methods.

3. PROBLEM DEFINITION

A boolean circuit is defined as a directed, acyclic graph $G(N, E)$ where N is a set of nodes and E is a set of edges. Each node $n_i \in N$ is either a primary input, an output or a logical gate, nodes are distinguished by their in and out degrees (see table 2); each node has a transition density $td(n_i)$ which is the number of times the signal changes in unit time. Each directed edge $e_i \in E$ connects one node to another. Figure 1A is an example of a boolean circuit. The total estimated average power consumption of a boolean circuit is shown in equation (1).

$$P_{avg}(B) = \begin{array}{l} \sum_{i=0}^{n_{in}} (\frac{1}{2} C_{in} V_{dd}^2 f_i td(n_i)) + \\ \sum_{i=n_{in}+1}^{n} (\frac{1}{2} (C_{in} + C_{out}) V_{dd}^2 td(n_i)) \end{array} \quad (1)$$

Where $P_{avg}(B)$ is the average power consumption of boolean circuit B, n_{in} is the number of input nodes, C_{in} and C_{out} are the circuit capacitances, V_{dd} is the circuit voltage, f_i is the fan out of input i and $td(n_i)$ is the transition density associated with node n_i.

A Look Up Table (LUT) circuit can be defined as a directed, acyclic graph $C(N', L, k, E')$ where N' is a set of nodes, L is a set of LUTs and E' is a set of edges. Each node $n_i \in N'$ is either a primary input node or an output node and has a transition density $td(n_i)$. Each $l_i \in L$ is an LUT; an LUT is defined as a special node which can represent any boolean function with up to k_{in} distinct inputs (where $k_{in} \geq 2$) and a single output i.e. an LUT node can replace a boolean circuit (or sub circuit) with no more than k_{in} input nodes and one output node. Every LUT has a transition density $td(l_i)$ which represents the switching activity of the LUT. Each directed edge $e_i \in E'$ connects one node or LUT to another node or LUT. The total estimated average power consumption of a LUT circuit is described in equation (2).

$$P_{avg}(L) = \begin{array}{l} \sum_{i=0}^{n} (\frac{1}{2} C_{in} V_{dd}^2 f_i td(n_i)) + \\ \sum_{i=0}^{l} (\frac{1}{2} (C_{in} + C_{out}) V_{dd}^2 td(l_i)) \end{array} \quad (2)$$

Where all terms are the same as in equation (1) with the following additions: L is an LUT based circuit and l is the total number of LUTs.

A boolean circuit can be implemented by a LUT circuit. A single LUT can represent any boolean circuit (or sub circuit) with k_{in} or less input nodes and one output node; a LUT circuit can therefore implement the same function as a boolean circuit by substituting sub circuits for LUTs. The average power consumption of the equivalent LUT circuit is equal to the boolean circuit which contains only the input nodes and output node of each sub circuit. Figure 1B shows an example of how boolean circuit 1A can be mapped onto a LUT circuit where $k_{in} = 4$ and the grey areas represent LUTs. A LUT circuit is considered to have mapped a boolean circuit if the LUT circuit implements the same function as the boolean circuit.

We wish to find a k feasible LUT mapping of any boolean circuit such that the average expected power consumption is minimised.

	In degree	Out degree
Primary input	0	≥ 1
Output	1	0
Logical Gate	2*	1

Table 2: In degree and out degree of node types

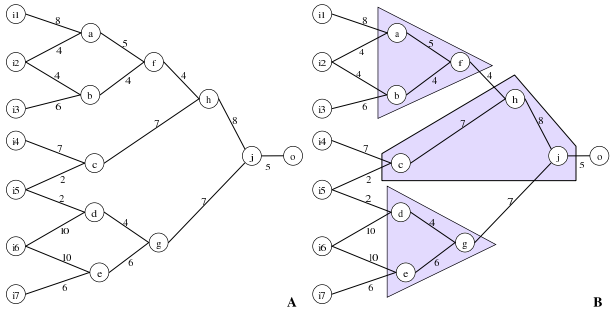

Figure 1: A: boolean circuit. B: LUT mapping of A.

4. SIMULATED ANNEALING

In this section we describe the simulated annealing algorithm which is applied to the problem described above. We begin by outlining a general homogeneous simulated annealing algorithm and a basic implementation of a simulated

*We state that the in degree of a logical gate is 2 and the out degree is 1. This does not restrict the power of the circuit as any unbounded gates can be decomposed into an equivalent boolean circuit with bounded gates [28]

annealing algorithm (Algorithm 1) which incorporates all of the features discussed in this section.

A simulated annealing algorithm attempts to find an optimal (or at least good approximate) solution to a hard problem by searching the solution space using local moves. Each time a local move is made the new solution is evaluated according to the objective function. If the new solution is better than the current solution it is accepted, if not the new solution is accepted according to the equation (3).

$$a = e^{\frac{f(c)-f(n)}{c(k)}} \tag{3}$$

Where a is the probability the solution will be accepted, $f()$ is the objective function, c is the current solution, n is the new solution and $c(k)$ is the current temperature.

Equation (3) relies on $c(k)$ which is defined as the current temperature but more accurately it is the temperature at step k (which in equation (3) is the current step). The temperature is defined in 2 parts, $c(0)$ which is the starting temperature and $c(k)$ which recursively defines the temperature at any step $k > 0$. $c(k)$ also defines the cooling schedule.

We have chosen to define $c(k)$ (equation (4)) using a cooling schedule which has been shown to have be very successful for similar problems. Literature suggests that $c(0)$ should be set such that even the worst possible move has a high probability of being accepted; this depends on the problem being considered.

$$c(k) = c(k-1) \cdot \frac{1}{b} \tag{4}$$

Where b is an algorithm parameter and k is the step of the algorithm.

The temperature is lowered at each iteration of the algorithm according to equation (4) until $c(k) < c(final)$ where $c(final)$ is the threshold. The threshold should be set such that there is a very low possibility that even the least bad move will be accepted; this again depends on the problem being considered.

As we are considering the homogeneous simulated annealing model we perform a number of local moves at each temperature rather than adjusting the temperature after each move. The length of the Markov chain (number of moves made and accepted) at each step is determined by equation (5).

$$L_c = h\eta \tag{5}$$

Where L_c is the number of moves accepted, h is an algorithm parameter η is the size of the neighborhood.

Finally we require a complete local move set and the neighborhood function. We define a complete moves set for our problem in the section below.

4.1 Move Set and Neighborhood

In this section we define a complete move set which allows us to transform one k_{in} feasible LUT mapping to any other k_{in} feasible LUT mapping through a series of local moves and describe a method to evaluate the energy consumption of the new covering.

4.1.1 Slightly Restricted Boolean Circuits

We begin by considering a slightly restricted version of a boolean circuit where each input node may have only one output, see table 3 (the reasoning behind this will become apparent later). We begin with some basic definitions.

Algorithm 1 Simulated Annealing

```
bestSol = currentSol = a random solution
k=0
while c(k) > c(final) do
  moveCount = 0
  while moveCount < L_c do
    newSol = LocalMove(currentSol)
    if newSol is accepted by equation (3) then
      moveCount++
      currentSol = newSol
      if f(currentSol) < f(bestSol) then
        bestSol = currentSol
      end if
    end if
  end while
  k++
end while
Return bestSolution
```

	In degree	Out degree
Primary input	0	1
Output	1	0
Logical Gate	2	1

Table 3: In degree and out degree of node types

Definition 1. *Every node in a boolean circuit has a flag which represents if the outgoing edge is cut or uncut; where all input nodes and the output node must be labeled 'cut'. The state of a boolean circuit is the list of node flags.*

Definition 2. *A k_{in} Feasible Partition is a boolean circuit which is divided such that each section can be implemented by a single k input LUT. A partition can be represented by a state.*

A boolean circuit with l logical gates has 2^l different states but not all of these correspond to a k_{in} feasible partition.

Local Move Set 1. *Given a boolean circuit and a state which represents a k_{in} feasible partition: pick a single node, which does not have to be cut (i.e. an input or output node) and invert the flag, this gives 2 distinct moves:*

- *Cut node: flip the flag on a node from uncut to cut*
- *Uncut node: flip the flag on a node from cut to uncut*

If the resulting state corresponds to a k_{in} feasible partition then the move is valid.

To make a valid move we can flip a random flag and then check the validity. We can check if the resulting state is valid in $O(l)$ time (where l is the number of logical gates) using a basic tree traversal but there is potential to improve the speed through multi-threading; this could potentially increase the speed by a factor of c (where c is the number of cut logical gate nodes) but the worst case bound remains the same.

Alternatively we can compute all possible moves which will result in a valid state in $O(l)$ time using a simple BFS

based algorithm. Each time we apply a move we can maintain the possible moves lists in worst case time $O(l)$; although in the majority of situations it it much quicker. In our implementation we have chosen to compute and maintain the list of valid moves as this only requires at most $O(l)$ computation time per accepted move and has been very efficient in practice.

4.1.2 Completeness

For the Local Move Set 1 to be complete over the set of restricted boolean circuits we need to show that any k_{in} feasible partition can be transformed into any other k_{in} feasible partition using a series of local moves. To simplify this problem we can consider the following sub problems:

1. Is the move set directly reversible?
2. can we use the local move set to transition from an initial k feasible partition p_0 where all nodes are cut to any other k_{in} feasible partition?

If we can prove that the both of the above are true then we have shown that we can move from any k_{in} feasible partition to any other k feasible partition.

Showing that move set 1 is reversible is trivial as cut and uncut are the exact opposite. We begin in k_{in} feasible partition s_x make a cut (or uncut) node move by flipping the flag on node n_i and transition to another k_{in} feasible partition s_y, if the new partition is not k_{in} feasible then the move is not valid. We know that by making the opposite uncut (or cut) node move by flipping the flag on node n_i we must transition back to the original partition s_x; hence move set 1 is directly reversible.

DEFINITION 3. *p_0 is the state where all nodes in a boolean circuit are cut, this corresponds to a k_{in} feasible partition where each node is implemented by a separate LUT.*

We begin by further simplifying the problem by considering any k_{in} feasible partition as a combination of boolean circuits with k_{in} $\geq$inputs where all logical gates are uncut, i.e. they form a single k_{in} feasible LUT. This means that without loss of generality we can just consider the situation where we take a boolean circuit with up to k_{in} inputs in state p_0 and use Move Set 1 to transform it into state p_1, this will prove that the move set is complete.

DEFINITION 4. *p_1 is the state where all logical gate nodes in a boolean circuit are uncut, if there are k_{in} or less inputs this corresponds to a k_{in} feasible partition where all nodes are implemented in a single LUT.*

LEMMA 1. *There exists a chain of 'uncut' moves (swapping an node from cut to uncut) which can transform a boolean circuit with k_{in} or less inputs from state p_0 into state p_1*

LEMMA 2. *A boolean circuit as defined in section 1 which has $n-1$ logical gates must have n input nodes and 1 output node*

The boolean circuit that we define in section 3 adhere to a tree structure and a tree with n leaves will have $n-1$ non leaf nodes where one is the root, therefore lemma 2 must be true. (Alternatively: if we have n input nodes and each logical gate has 2 inputs and 1 output then we will require $n-1$ logical gates to connect all of the inputs.)

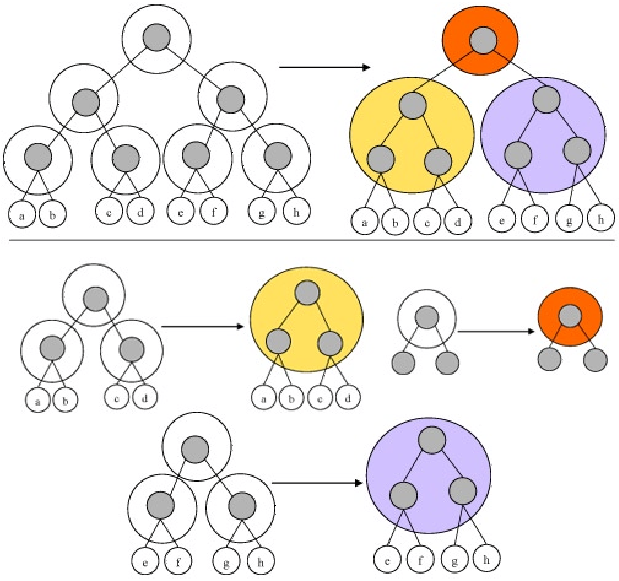

Figure 2: A diagram which shows how a complex partition can be considered a combination of p_1 partitions

Lemma 2 shows that the number of inputs is connected to the number of nodes; we know if the maximum number of input nodes allowed is less than or equal to k_{in} then we can have at most $k_{in}-1$ logical gates. If the maximum number of logical gates is $k_{in}-1$ then any intermediate states must have less then $k_{in}-1$ logical gates and therefore less than k_{in} inputs. i.e. a sub tree cannot have more leaves than the original tree. We can therefore use the uncut move to transition a boolean circuit from state p_0 to state p_1.

We have proved that we can transition a boolean circuit from state p_0 to state p_1 using move set 1 and that move set 1 is reversible, hence we have proved that the move set 1 must be complete as it allows us to access all possible k feasible partitions.

4.1.3 Unrestricted Boolean Circuit

We have proved in Section 4.1.2 that Move Set 1 is complete over our initial restricted definition of a boolean circuit. We now consider the original boolean circuit definition where each input node has unrestricted fan out.

Move Set 1 is not complete over the unrestricted definition of a boolean circuit. This is because Lemma 2 cannot hold as nodes can now share input nodes. Therefore a partition may have more than $k-1$ nodes. To account for this we include 2 additional moves which are defined in Local Move Set 2.

LOCAL MOVE SET 2. *Given a boolean circuit and a state which represents a k_{in} feasible partitioning: pick both children of a node, if the children are not marked that they should always be cut and are either both cut or both uncut then we can invert both flags, this gives 2 distinct moves:*

- *Cut Children: flip the flag on the nodes from uncut to cut*

- *Uncut Children: flip the flag on the nodes from cut to uncut*

If the resulting state corresponds to a k_{in} feasible partition then the move is valid.

We can use an extended BFS based algorithm to produce a list of all valid moves in $O(l)$ time. In the following section when we refer to Local Move Set 2 we are implicitly referring to any move which is valid over Local Move Set 1 or 2.

4.1.4 Completeness

In order to prove the completeness of Move Set 2 we need to show that:

- the move set is reversible?
- the move set works in situations where Lemma 2 does not hold?

In the above section we showed that the cut and uncut node moves were reversible very simply. This case is very similar as we are just applying 2 cut node moves at the same time and hence the logic remains the same. We can only use a cut (or uncut) children move if the resulting partition is k_{in} feasible and by applying the opposite move we transition back into the original k_{in} feasible partition. Hence cut and uncut children are reversible.

For Lemma 2 to not hold the number of nodes must be greater than or equal to the number of inputs. This can only happen when at least one input node has a fan out greater than 1 and more than one of the fan out edges are inputs of the same LUT.

DEFINITION 5. *S is a set of boolean circuits in a k_{in} feasible partition where at least one section (LUT) has k_{in} input nodes and at least k_{in} logical gates, i.e. the set of k_{in} feasible partitions where Lemma 2 does not hold.*

LEMMA 3. *There exists a boolean circuit $b_1 \in S$ in state s_1 which corresponds to a k feasible partition with the set of input nodes I, the set of logical gates L and a single output node o where $|I| \geq |L|$.*

We know that Lemma 3 must be true as Figure 3.4 is an example of this.

LEMMA 4. *There exists a boolean circuit b_1 in a k feasible partition s_1 from lemma 3. If both child nodes of the output node o are cut then the resulting partition will also be k_{in} feasible.*

The children of o (which we label o_l and o_r) may be input nodes or logical gates; this gives 3 distinct possibilities:

1. o_l is an input node and o_r is a logical gate
2. o_l is a logical gate and o_r is an input node.
3. Both o_l and o_r are logical gates

Note that we have omitted the case where both o_{left} and o_{right} are input nodes as this would require there to be only one logical gate (which is also the output node) and for lemma 2 not to hold then either $k_{in} < 2$ or both inputs must be the same which renders the logical gate redundant.

Case 1 and 2 are equivalent as they both contain one input node and one logical gate. The description of a boolean circuit states that an input node must be cut so only the logical gate needs to be cut and therefore we apply the cut node move. By cutting the logical gate node we are left with a new partition with 2 sections; the first contains only one node o and the second contains all other logical gates. The second section (containing all logical gates apart from o) has an input set $I' \subseteq I$ therefore the number of inputs can be at most k_{in} and the partition is k_{in} feasible.

In case 3 both children are logical gates and hence they both need to be cut and we must apply the cut children move. Once both cuts are applied we reach a new partition with 3 sections; the first contains only o, the second contains the sub circuit attached to o_l and the third contains the sub circuit attached to o_r. The first section can have only 2 inputs and hence must be k_{in} feasible. The second and third may have between 1 and $|L| - 1$ logical gates in each sub circuit, each section has an input set which we will label I_l and I_r where $I_l \subseteq I$ and $I_r \subseteq I$ and $I_l \cup I_r = I$. As each section may have at most $|I|$ inputs the partition must be k_{in} feasible.

We have shown that all cases lead to a new k_{in} feasible partition and hence lemma 4 must be true. We can now apply lemma 4 recursively to decompose a circuit from partition s_1 to partition s_0 (as is shown in figure 3); as we have shown that the move set is reversible we can state that the move set is complete.

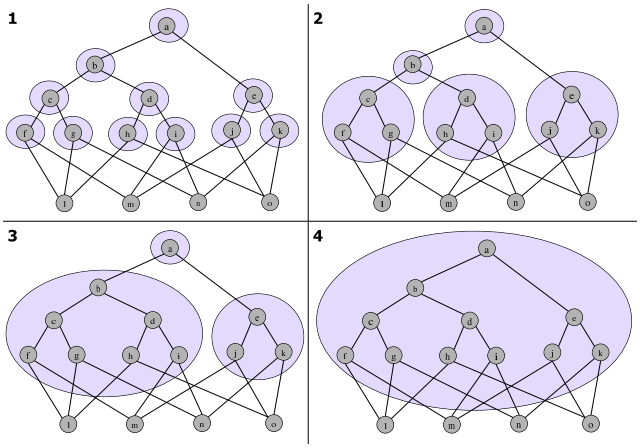

Figure 3: Repeated application of Local Move Set 2

4.2 Simulated Annealing Parameters

In this section we formally define the parameters used in our simulated annealing algorithm.

The starting temperature is set such that any move should be accepted with a high probability. We calculate this by rearranging the acceptance probability to make the $c(0)$ the object:

$$a = e^{\frac{f(c)-f(n)}{c(k)}} \quad \rightarrow \quad c(0) = \frac{\Delta^w}{\ln p} \tag{6}$$

Where Δ^w is the increase in the objective function when the worst possible move is made (which can be simply computed in $O(l)$ time before the simulated annealing algorithm begins) and p is the probability which the move should be accepted with. When p is set to 0.999 (i.e. $\leq 0.01\%$ chance that the move would not be accepted) the equation becomes:

$$c(0) = 1000\Delta^w \tag{7}$$

The algorithm stops looping once the temperature drops below a threshold at which point it is unlikely (< 0.05) that even the least bad move would be accepted. The definition of the problem is such that the least bad move is never very bad so requiring that this has less than a 5% chance of being accepted gives the algorithm ample time to ensure the algorithm has not halted during a gradient decent. This again is defined by rearranging the acceptance criteria equation:

$$a = e^{\frac{f(c)-f(n)}{c(k)}} \quad \rightarrow \quad c(final) = \frac{\Delta^l}{\ln p'} \tag{8}$$

Where $c(final)$ is the threshold, p' is the probability that the current solution will be accepted and Δ^l is the increase in cost cause by the least bad move possible. When we set p' to 0.05 we get the following equation:

$$c(final) = 0.33381\Delta^l \tag{9}$$

In the general simulated annealing definition above we define the $c(k)$ (equation (4)) and L_c (equation (5)) both of which take additional parameters. The parameter b from equation (4) is set to be a small number such that the cooling is slow, for our tests we experimentally set $b = 2$. The parameter h from equation (5) is set such that the number of moves at a given temperature is sufficiently large, for our tests we experimentally set $h = 20$.

4.2.1 Implementation and Analysis

Our simulated annealing algorithm can be implemented very efficiently which is essential due to the number iterations the algorithm requires. We outline the running times of various parts of our simulated annealing algorithm in Algorithm 2. The only possible point of contention is that it takes up to $O(n)$ time to update the list of possible moves, our analysis shows that in the majority of cases this can be achieved in small constant time but for small circuits it is possible that this takes $O(n)$ as one local move can have ramifications for the entire circuit. When this is the case it is highly likely that the the value of n is sufficiently small that the algorithm can still compute all possible moves in a very short time.

Algorithm 2 Simulated Annealing Analysis

```
Generate random solution - O(n)
Evaluate initial solution - O(n)
Calculate all possible moves - O(n)
while temperature > threshold do
  while movesMade < L_c do
    Pick random move - O(1)
    Generate new solution - O(1)
    Evaluate new solution and possibly accept - O(1)
    Update list of possible moves - O(n)
  end while
end while
Return Best Solution
```

5. RESULTS

For our experiments we implemented the Simulated Annealing algorithm using Python and tested with a combination of randomly generated and MCNC benchmark boolean circuits. We utilized the MVSIS [4] *strash* command to convert the MCNC circuits (in blif format) into 2 bounded AND2 & INVERTER boolean circuits which were then saved as '.bench' files.

Circuit	SIS	GA	SA	Percentage	
Misex2	3.59	2.85	2.37	66.01%	83.16%
Sao2	8.36	7.24	5.22	62.44%	72.10%
Con1	1.53	1.19	0.49	32.02%	41.17%
5xp1	2.24	1.71	1.24	55.36%	72.51%
Rd53	4.32	3.05	2.50	57.87%	81.97%
Z4ml	6.98	5.56	4.72	67.62%	84.89%
			Average	57.04%	72.56%

Table 4: Results Comparison

In order to compare our results with those from Pandey et. al. and SIS we present our findings in terms of cumulative switching; which is directly related to power consumption of a circuit. The switching of a signal calculated using equation (10) which is quoted from [20].

$$2 \cdot p(s) \cdot (1 - p(s)) \tag{10}$$

Where $p(s)$ is the probability of signal s begin 1.

We initialized our simulated annealing algorithm with $b = 2$ and $h = 20$ and map the input boolean circuits onto LUTs with $k_{in} = 5$. Each test is ran once until completion and the best found solution is reported. The genetic algorithm from [20] took an average of 3.2 seconds to produce the results in Table 4. Our simulated annealing algorithm used considerably more time to produce our results ($2\frac{1}{4}$ hours for 5xp1) but they provide such a great reduction in power consumption that we consider this to be a reasonable time cost.

In table 4 we report the results from our experiments along side the results for SIS and the Genetic algorithm quoted in [20]. We can see that as expected our SA based algorithm has the ability to severely reduce the power consumption of SIS; in the case of circuit Con1 by 67.98% and by an average of 42.96%. Furthermore we see that the SA algorithm has reduced the power consumption of the genetic algorithm by an average of 27.44% and in the case of Con1 by 58.83%.

6. CONCLUSIONS

In this paper we have formally defined the problem of low power LUT FPGA mapping as a combinatorial optimisation problem. We introduced a local move set which has been shown to be complete for traversing all solutions in our problem definition. Experimental results with our proposed simulated annealing procedure have been compared to two alternative approaches and demonstrate that the SA algorithm can produce better results than both. Most notably our results decrease the cumulative switching (which is analogous to power consumption) by up to 27.44%. These results motivate further investigations of simulated annealing in combination with more tailored cooling schedules. In future research we plan to analyse the convergence properties of SA for the problem and to implement a wide range of alternative local search algorithms for a comprehensive comparison.

7. REFERENCES

[1] E. H. Aarts, J. H. Korst, and P. J. Van Laarhoven. Simulated annealing. *Local search in combinatorial optimization*, pages 91–120, 1997.

[2] J. Anderson and F. Najm. Power-aware technology mapping for lut-based fpgas. In *Field-Programmable Technology, 2002. Proceedings. 2002 IEEE International Conference on*, pages 211–218, 2002.

[3] G. Boole. *An Investigation of the Laws of Thought: On which are Founded the Mathematical Theories of Logic and Probabilities*. Walton and Maberly, 1854.

[4] R. K. Brayton and S. P. Khatri. Multi-valued logic synthesis. In *VLSI Design, 1999. Proceedings. Twelfth International Conference On*, pages 196–205. IEEE, 1999.

[5] I. Bucur, N. Cupcea, A. Surpateanu, C. Stefanescu, and F. Radulescu. Power-aware, depth-optimum and area minimization mapping of k-lut based fpga circuits. *WSEAS Transactions on Computers*, 8(11):1812–1824, 2009.

[6] V. Ĉerny. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of Optimization Theory and Applications*, 45:41–51, 1985.

[7] C.-S. Chen, T. Hwang, and C. L. Liu. Low power fpga design - a re-engineering approach. In *Proceedings of the 34th annual Design Automation Conference*, DAC '97, pages 656–661, New York, NY, USA, 1997. ACM.

[8] J. Chen, X. Wei, Q. Zhou, and Y. Cai. Power optimization through edge reduction in lut-based fpga technology mapping. In *Communications, Circuits and Systems, 2009. ICCCAS 2009. International Conference on*, pages 1087–1091, 2009.

[9] A. Farrahi and M. Sarrafzadeh. Complexity of the lookup-table minimization problem for fpga technology mapping. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 13(11):1319–1332, 1994.

[10] A. Farrahi and M. Sarrafzadeh. Fpga technology mapping for power minimization. In R. Hartenstein and M. Servit, editors, *Field-Programmable Logic Architectures, Synthesis and Applications*, volume 849 of *Lecture Notes in Computer Science*, pages 66–77. Springer Berlin / Heidelberg, 1994.

[11] R. H. Freeman. Configurable electrical circuit having configurable logic elements and configurable interconnects, sep 1989.

[12] J.-M. Hwang, F.-Y. Chiang, and T. Hwang. A re-engineering approach to low power fpga design using spfd. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 722–725, New York, NY, USA, 1998. ACM.

[13] S. Kirkpatrick, M. Vecchi, et al. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[14] B. Kumthekar, L. Benini, E. Macii, and F. Somenzi. In-place power optimization for lut-based fpgas. In *Proceedings of the 35th annual Design Automation Conference*, DAC '98, pages 718–721, New York, NY, USA, 1998. ACM.

[15] B. Kumthekar, L. Benini, E. Macii, and F. Somenzi. Power optimisation of fpga-based designs without rewiring. In *Computers and Digital Techniques, IEE Proceedings-*, volume 147, pages 167–174. IET, 2000.

[16] H. Li, S. Katkoori, and W.-K. Mak. Power minimization algorithms for lut-based fpga technology mapping. *ACM Trans. Des. Autom. Electron. Syst.*, 9(1):33–51, 2004.

[17] H. Li, W.-K. Mak, and S. Katkoori. Lut-based fpga technology mapping for power minimization with optimal depth. In *VLSI, 2001. Proceedings. IEEE Computer Society Workshop on*, pages 123 –128, 2001.

[18] H. Li, W.-K. Mak, and S. Katkoori. Efficient lut-based fpga technology mapping for power minimization. In *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pages 353–358, 2003.

[19] M. Mashayekhi, Z. Jeddi, and E. Amini. Power optimization of lut based fpga circuits. In *Optimization of Electrical and Electronic Equipment, 2008. OPTIM 2008. 11th International Conference on*, pages 37–40, 2008.

[20] R. Pandey and S. Chattopadhyay. Low power technology mapping for lut based fpga - a genetic algorithm approach. In *VLSI Design, 2003. Proceedings. 16th International Conference on*, pages 79–84, 2003.

[21] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sis: A system for sequential circuit synthesis. 1992.

[22] C. Shannon. A symbolic analysis of relay and switching circuits. *American Institute of Electrical Engineers, Transactions of the*, 57(12):713–723, 1937.

[23] K. Steinhöfel, A. Albrecht, and C. Wong. Two simulated annealing-based heuristics for the job shop scheduling problem. *European Journal of Operational Research*, 118(3):524–548, 1999.

[24] K. O. Tinmaung, D. Howland, and R. Tessier. Power-aware fpga logic synthesis using binary decision diagrams. In *Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays*, FPGA '07, pages 148–155, New York, NY, USA, 2007. ACM.

[25] T. Tuan, A. Rahman, S. Das, S. Trimberger, and S. Kao. A 90-nm low-power fpga for battery-powered applications. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 26(2):296–300, 2007.

[26] C.-C. Wang and C.-P. Kwan. Low power technology mapping by hiding high-transition paths in invisible edges for lut-based fpgas. In *Circuits and Systems, 1997. ISCAS '97., Proceedings of 1997 IEEE International Symposium on*, volume 3, pages 1536 –1539, 1997.

[27] Z.-H. Wang, E.-C. Liu, J. Lai, and T.-C. Wang. Power minimization in lut-based fpga technology mapping. In *Design Automation Conference, 2001. Proceedings of the ASP-DAC 2001. Asia and South Pacific*, pages 635 –640, 2001.

[28] I. Wegener. *The complexity of Boolean functions*. Eiley-Teubner, 1987.

[29] X. Wei, J. Chen, Q. Zhou, Y. Cai, J. Bian, and X. Hong. Macromap: A technology mapping algorithm for heterogeneous fpgas with effective area estimation. In *Field Programmable Logic and Applications, 2008. FPL 2008. International Conference on*, pages 559–562. IEEE, 2008.