Automating RT-Level Operand Isolation to Minimize Power Consumption in Datapaths

M. Münch[1], B. Wurth[2], R. Mehra[3], J. Sproch[3], and N. Wehn[1]

[1] University of Kaiserslautern
Erwin-Schroedinger-Strasse
D-67663 Kaiserslautern, Germany
{*michaelm, wehn*}*@eit.uni-kl.de*

[2] Infineon Technologies AG
P.O.Box 800949
D-81609 Munich
bernd.wurth@infineon.com

[3] Synopsys, Inc.
700 East Middlefield Rd.
Mountain View, CA 94043
{*renu, jims*}*@synopsys.com*

Abstract

Designs which do not fully utilize their arithmetic datapath components typically exhibit a significant overhead in power consumption. Whenever a module performs an operation whose result is not used in the downstream circuit, power is being consumed for an otherwise redundant *computation. Operand isolation [3] is a technique to minimize the power overhead incurred by redundant operations by selectively blocking the propagation of switching activity through the circuit.*

This paper discusses how redundant operations can be identified concurrently to normal circuit operation, and presents a model to estimate the power savings that can be obtained by isolation of selected modules at the register-transfer (RT) level. Based on this model, an algorithm is presented to iteratively isolate modules while minimizing the cost incurred by RTL operand isolation. Experimental results with power reductions of up to 30% *demonstrate the effectiveness of the approach.*

1 Introduction

In certain classes of designs, arithmetic datapath components are used only occasionally and spend a large amount of time in an idle state. The most prominent examples are control-dominated designs with arithmetic operations that are used only in a few states, precluding their full utilization. Other examples include re-used designs of which only part of the functionality is being used. These designs are often characterized by a significant overhead in power consumption. Whenever a module performs an operation whose result is not used in the downstream circuit, it unnecessarily consumes power. We will call such an operation *redundant*.

The idea of *operand isolation* is to identify redundant operations and, using special isolation circuitry, prevent switching activity from propagating into a module whenever it is about to perform a redundant operation. Therefore, the transition activity of the internal nodes of the module and, to a certain extent, its transitive fanout is reduced significantly, resulting in lower power consumption [2].

To illustrate the concept of operand isolation, let us consider a small example. Figure 1 shows a part of a design in which the result of an operation performed by adder a_0 is evaluated conditionally in its transitive fanout. For certain configurations of the multiplexor select signals S_0, S_1, and S_2 and the register load enable signals G_0 and G_1, the output of a_0 is not used to compute the values to be stored in registers r_0 and r_1. However, a_0 will continue to compute a new output whenever there is switching activity at its inputs *A* and *B*, therefore consuming power by executing *redundant computations*. For long periods in which the output is not used, this power overhead can be substantial.

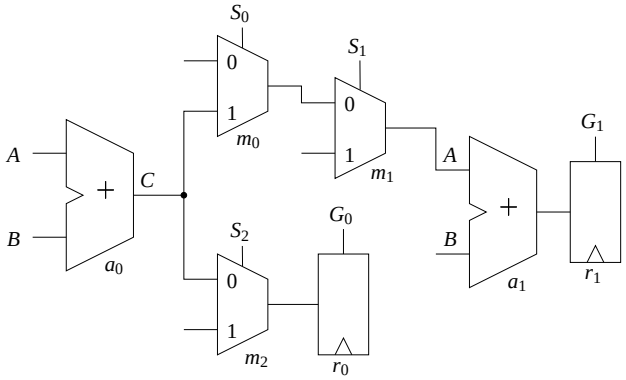

Figure 1: Design without operand isolation.

Suppose now that there is an *activation signal* AS_{a_0} whose logic value indicates if a_0 performs a computation

that is not redundant. We can use AS_{a_0} to control blocking logic, e.g. transparent latches that "freeze" the inputs of a_0, effectively preventing the propagation of switching activity into the module. The module will therefore only perform non-redundant computations. The lower transition probability at the internal nodes of the module will then result in lower power consumption. Figure 2 shows the same circuit where the inputs of the two adders have been isolated using latches. Assuming that AS_{a_0} evaluates logic '0' whenever a_0 is performing a redundant computation, inputs A' and B' maintain their previous values and do not transition when the operation to be performed by a_0 is redundant.

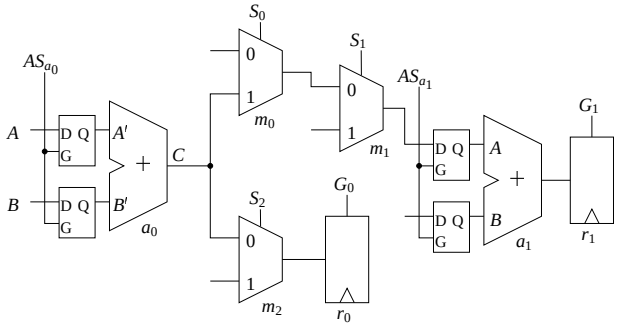

Figure 2: Design with operand isolation.

An algorithm to automate the application of operand isolation to a given RTL circuit will have to solve the following two key problems:

- Select a set of modules for which operand isolation results in the largest reduction in power consumption for the overall circuit, and
- for each of these modules, obtain an activation signal which indicates that the module is performing a computation that is not redundant.

This paper makes contributions in both areas to obtain the first comprehensive approach that automates operand isolation on RT level. After discussing related research in Section 2, we will present a method to derive an activation signal for a module in Section 3. In Section 4 we will present a model to estimate the power savings that can be obtained by isolating a module. This model is the basis of the iterative algorithm presented in Section 5. While trading off power savings vs. the cost in terms of power, area, and timing associated with isolation logic, the algorithm iteratively isolates modules to obtain a power-optimized RTL circuit with least overhead. Results which demonstrate the efficiency of the approach are presented in Section 6.

2 Related Work

Previous work as well as our approach exploit *observability don't care* conditions to temporarily stop the propagation of switching activity through logic blocks. The various approaches differ w.r.t. the circuit's abstraction level (gate or RTL), the activation signal, and the circuitry blocking the propagation of switching activity.

To the best of our knowledge, operand isolation was first presented in literature in [3]. It was successfully used to minimize power in the datapath of the IBM PowerPC 4xx family of embedded controllers. However, the technique was applied manually and with only a local scope: the paper describes the isolation of modules feeding multiplexors, where the multiplexor select signal is used as the activation signal.

TIWARI *et al*. use the term "*guarded evaluation*" for what is essentially operand isolation on the gate level [9]. While guarded evaluation can isolate arbitrary combinational logic blocks, this also means that it must identify appropriate points at which guard logic can be inserted. In an RTL operand isolation approach as proposed in this paper, these points are naturally given by the inputs of arithmetic modules. Working at the RT level provides the additional advantage that RTL structural information can be used to efficiently generate activation signals and the computationally expensive implication checking used to implement *guarded evaluation* is not required. Moreover, guarded evaluation uses an existing signal from the circuit as activation signal. The existence of such a signal, however, cannot be guaranteed.

In [4], KAPADIA *et al*. present a technique to reduce power consumption on datapath busses by stopping the propagation of switching activity through bus-driver modules, which is effectively an RTL operand isolation technique. As in our approach, gating signals (activation signals) are constructed from existing control signals, which provides broad isolation coverage. A key idea in [4] is to avoid the insertion of activity-blocking transparent latches; switching activity is blocked by gating enable signals of registers. As a consequence, modules driven by multiple-fanout registers cannot be optimally isolated (Fig. 7 in [4]). Also, no power savings are possible in combinational logic that is directly fed by primary inputs. No models to estimate potential power savings are presented.

3 Identifying Redundant Computations

As stated in Section 1, one of the key issues in operand isolation is the availability of a signal to indicate that a module is performing a computation that is not redundant. In [9] such a signal is extracted from the logic description of the circuit based on the *observability don't care* conditions of

the module to be isolated. Unfortunately, we cannot guarantee the existence of such a signal which covers all relevant isolation cases in the circuit. Moreover, even if it exists in the logic description of the circuit, it might not be visible in an RT structure since it might be hidden in a complex RT module. Therefore, rather than using an existing signal, we will compute an *activation function*, f, which indicates an isolation case. This computation is done by a structural analysis of the transitive fanout of a module. This function is implemented by the *activation logic* which is either a direct implementation or an optimized version thereof. The activation logic generates the *activation signal*, *AS*, used to control the *isolation banks*, i.e., the set of gates used at the module inputs to block input transitions during redundant operations.

To illustrate the derivation of the activation function, let us again consider the circuit in Figure 1. For both adders a_0 and a_1 we are looking for activation functions f_{a_0} and f_{a_1}, respectively, which evaluate '0' whenever the adder is performing a redundant computation. Adder a_1 is performing a non-redundant computation if its output is stored in register r_1 (i.e. $G_1 =$ '1') *and* the value stored in r_1 will be evaluated in the transitive fanout of r_1 in the *subsequent* clock cycle. Activation function f_{a_1} therefore depends on the value $f_{r_1}^+$ of the activation function f_{r_1} of register r_1 in the following clock cycle (as indicated by the superscript '$+$'):

$$f_{a_1} = G_1 f_{r_1}^+.$$

Likewise, a_0 performs a non-redundant computation whenever its output is stored in r_0 *and* the value in r_0 will be evaluated in its transitive fanout in the subsequent clock cycle *or* is being used as input to a_1 *and* a_1 is performing a non-redundant computation. Whether or not the output of a_0 is observable at the inputs of a_1 and r_0 depends on the values of the select inputs to m_0, m_1, and m_2. We can now formulate f_{a_0} as follows:

$$\begin{aligned} f_{a_0} &= \overline{S_2} G_0 f_{r_0}^+ + S_0 \overline{S_1} f_{a_1} \\ &= \overline{S_2} G_0 f_{r_0}^+ + S_0 \overline{S_1} G_1 f_{r_1}^+. \end{aligned}$$

A similar strategy can be used for RT structures which use arbitrary logic in conjunction with modules, multiplexors and registers. Any gate can be interpreted as a degenerated multiplexor, where the Boolean function which specifies when a change at an input to the gate is observable at its output can be derived based upon its *controlling value* [1].

The example shows that by using a breadth-first traversal starting at the primary outputs of a circuit, we can compute in $O(|V| + |E|)$ time an activation function for each arithmetic module, where V is the set of nodes representing modules and E the set of edges representing nets in the RT network graph. Two problems arise when computing activation functions based on the entire transitive fanout of a module. With increasing depth of a module's transitive fanout, the corresponding activation function will grow more complex. An implementation of such a function will therefore incur a large area, timing, and power overhead. This may even offset the reduction in power dissipation obtained by operand isolation. Moreover, some of the signals originating "deep" in the transitive fanout may only occasionally cause the activation funtion to evaluate '0', therefore causing more overhead than benefit.

A more difficult problem is that the above strategy requires a "look-ahead" to pre-compute signal values in subsequent clock cycles when traversing sequential elements. Suppose, for instance, that the output of r_1 is connected to a 2:1 multiplexor with select signal S_3. The activation function of r_1 will therefore depend on S_3. However, only the value of S_3 as evaluated during the lifetime of r_1 starting in the following clock cycle will determine whether the output of a_1 will be propagated through the multiplexor. To compute f_{a_1} we therefore have to be able to pre-compute the value of S_3 one clock cycle in advance. There are essentially two ways to do so: Either by a structural analysis of the fanin of S_3 or by analyzing the corresponding FSM which computes the value of S_3. In practice, however, the resulting logic can also depend on primary inputs to the circuit, whose values obviously cannot be predicted. To make sure that an activation function can *always* be derived and to avoid the computational complexity associated with an FSM analysis, we define the activation function of a register to be constant '1'. By doing so, we effectively exclude isolation cases stemming from the fanout of sequential elements, but considerably reduce the computational complexity of deriving the activation functions as well as the complexity of the resulting circuitry to generate the activation signal. For the example in Figure 2, we therefore obtain the following simplified activation signals:

$$\begin{aligned} AS_{a_1} &= G_1 \\ AS_{a_0} &= \overline{S_2} G_0 + S_0 \overline{S_1} G_1. \end{aligned}$$

The assumption $f_r^+ = 1$ for a register r allows us to compute the activation functions locally in each combinational logic block bounded by sequential elements and primary inputs and outputs using the above-mentioned breadth-first technique.

4 Estimating Power Savings

In the following, we assume that for a given set of modules $\mathcal{C}$ called *isolation candidates* the corresponding activation functions have been computed based on the technique outlined in the preceding section. The isolation candidates are complex arithmetic operators for which operand isolation is expected to have a significant impact on the overall power consumption. To simplify the following discussion, we will generally consider two-input isolation candidates

with inputs A and B and a single output C—the models, however, are equally valid for multi-input and multi-output modules with a straightforward extension. Before deriving a model to estimate the power reduction obtained by isolating a candidate, we shall discuss what parameters affect the power reduction.

4.1 Parameters Affecting Power Reduction

The power consumption of a module can be characterized as a function of the toggle rates at its inputs using so-called *macro power models* [5, 7]. The toggle rate of a signal is the average number of toggles per clock cycle measured during a simulation of "real-life" test vectors. We assume that for each $c_i \in \mathcal{C}$ such a macro power model $p_i(\mathbf{Tr})$ as a function of a vector $\mathbf{Tr}$ of input toggle rates is available. For an isolation candidate c_i the power consumption $P(c_i)$ is therefore

$$P(c_i) = p_i(\mathrm{Tr}_A(c_i), \mathrm{Tr}_B(c_i)),$$

where $\mathrm{Tr}_A(c_i)$ and $\mathrm{Tr}_B(c_i)$ are the average toggle rates at inputs A and B, respectively.

The toggle rate at an input A is determined by the fanin logic feeding the input, consisting of combinational logic, and bounded by sequential elements, primary inputs and outputs, and other isolation candidates. We denote this fanin logic network as $L_A(c_i)$. In an RT structure such as the one in Figure 1, the fanin logic is a logic network made up of multiplexors and generic logic gates. This logic network is in turn fed by other isolation candidates, registers or primary inputs of the circuit. We call the set of isolation candidates which are connected to an input A of c_i via a combinational logic network, the *fanin candidates* of A and denote this set by $\mathcal{C}_A^-(c_i)$. In Figure 1, adder a_0 is a fanin candidate of input A of a_1, i.e. $a_0 \in \mathcal{C}_A^-(a_1)$. Likewise, we define a set a *fanout candidates* $\mathcal{C}_C^+(c_i)$. For the example we have $a_1 \in \mathcal{C}_C^+(a_0)$.

Apparently, the toggle rate $\mathrm{Tr}_A(c_i)$ measured for A is a function of the toggle rates at the outputs of the fanin candidates $\mathcal{C}_A^-(c_i)$ (for the sake of simplicity, we will neglect primary inputs). However, $\mathrm{Tr}_A(c_i)$ will not be identical to the toggle rate at the output of exactly one fanin candidate, since the logic network $L_A(c_i)$ will connect different fanin candidates to A depending on the signals controlling the configuration of $L_A(c_i)$[1]. For each $c_k \in \mathcal{C}_A^-(c_i)$, we therefore define a Boolean *multiplexing function* $g_{i,A}^k(\mathbf{x})$ which evaluates '1' if and only if $L_A(c_i)$ is configured such that c_k is connected to input A of c_i. The Boolean vector $\mathbf{x}$ is the vector of signals controlling the configuration of $L_A(c_i)$—the multiplexing function can be derived via a traversal of $L_A(c_i)$. In our example, we obtain $g_{a_1,A}^{a_0} = S_0\overline{S_1}$. It is now easy to see that $\mathrm{Tr}_A(c_i)$ is also a function of $g_{i,A}^k(\mathbf{x})$.

To summarize, the following parameters determine the power consumption of an isolation candidate

1. The set of fanin candidates $\mathcal{C}_A^-(c_i)$ and $\mathcal{C}_B^-(c_i)$ for both inputs A and B as well as the toggle rates at the outputs of the fanin candidates.

2. The multiplexing functions $g_{i,A}^k(\mathbf{x})$ and $g_{i,B}^j(\mathbf{x})$ which represent the behavior of the input logic networks $L_A(c_i)$ and $L_B(c_i)$, respectively.

Going back to Figure 2 with the observations above we note that isolating a_0 not only saves power by reducing the transition probability inside of a_0 itself, but also by reducing the toggle rate at input A of a_1. Since the output of a_0 is held quiescent whenever $AS_{a_0} = 0$, the toggle rate at input A is zero if $g_{a_1,A}^{a_0} = 1$, reducing the power consumption of a_1 accordingly. We therefore distinguish two types of power savings:

- *Primary savings* is the power saved in the isolation candidate c_i itself.

- *Secondary savings* is the power saved in the fanout candidates $\mathcal{C}_C^+(c_i)$.

We will now discuss the general ideas that underly a model to estimate primary and secondary savings separately. Rather than presenting the model for the general case, we illustrate the concepts via the example in Figure 1. A detailed discussion of the general case can be found in [6].

4.2 Primary Power Savings

In a first approximation assuming an even distribution of the toggle rate over the entire simulation interval, the primary power savings $\Delta P_p(c_i)$ obtained by isolating an isolation candidate c_i is proportional to the amount of time it spends performing redundant computations and the power used per computation, i.e.

$$\Delta P_p(c_i) = \Pr(\overline{f_{c_i}}) \cdot p_i(\mathrm{Tr}_A(c_i), \mathrm{Tr}_B(c_i)), \qquad (1)$$

where $\Pr(\overline{f_{c_i}})$ is the probability that the activation function f_{c_i} of c_i evaluates '0'. This probability is also computed during simulation.

The model errs when the input toggle rates are in fact not evenly distributed over the simulation interval. This is typically the case whenever the isolation of a module affects the toggle rate at the input of another module. For instance, in Figure 2 isolation of a_0 affects the toggle rate of input A of a_1. The output toggle rate $\mathrm{Tr}_C(a_0)$ is zero during intervals where $AS_{a_0} = 0$. Since the toggle rate *averages* over the

[1] We assume that the additional toggling induced by $L_A(c_i)$ is negligible.

entire simulation period, this means that the actual toggle rate when $AS_{a_0} = 1$ is higher than the one measured.

In Section 3 we have already made the assumption that isolation of candidates across sequential boundaries does not affect each other. The number of isolation candidates within a combinational region bounded by sequential cells and/or primary inputs and outputs is typically very small; moreover secondary power savings are likely to be significant. This means that it is reasonable to isolate one isolation candidate in each combinational block at a time in an iterative fashion (cf. Section 5). This also makes the problem of modeling inter-dependencies between different isolation candidates much easier to solve, since the toggle rate at the output of a candidate *after* isolation can be measured by simulation in the following iteration. The actual toggle rate, i.e. the toggle rate at the output of the isolated module during non-redundant computation cycles can then be derived from the measured toggle rate by simply scaling with respect to the actual number of cycles the candidate has been performing non-redundant operations.

For our example, we therefore obtain the actual toggle rate $\mathrm{Tr}'_C(a_0)$ observed at the output C of a_0 during non-redundant cycles as follows:

$$\mathrm{Tr}'_C(a_0) = \frac{\mathrm{Tr}_C(a_0)}{\Pr(AS_{a_0})}. \tag{2}$$

This has proven to be a good approximation of the actual toggle rate measured only during non-redundant computations. Taking into account the configuration of the two multiplexors m_0 and m_1 the share of power consumption of a_1 caused by the output toggle rate of a_0 is

$$\begin{aligned}\Pr(AS_{a_1} AS_{a_0} g^{a_0}_{a_1,A}) \cdot p_{a_1}(\mathrm{Tr}'_C(a_0), \cdot) \\ + \Pr(AS_{a_1} \overline{AS_{a_0}} g^{a_0}_{a_1,A}) \cdot p_{a_1}(0, \cdot),\end{aligned} \tag{3}$$

when neglecting the toggle rate at input B (as indicated by the dot '$\cdot$'). The first term is the power consumption due to the output toggle rate of a_0 when a_0 is performing a non-redundant computation, the second term the power consumption due to a_0 when it is performing a redundant computation. Note that the probabilities cannot further be simplified, since we cannot assume statistical independence of the various activation and multiplexing signals.

To obtain a general expression for the primary power savings $\Delta P_p(c_i)$, we have to consider each pair $(c_j, c_k) \in \mathcal{C}^-_A(c_i) \times \mathcal{C}^-_B(c_i)$ of fanin candidates and its probability to be connected to the inputs of c_i. This results in a formula similar to Eq. (3), where $\Delta P_p(c_i)$ includes four terms for each pair of fanin candidates, one for each situation in which c_j and c_k perform redundant and non-redundant computations, respectively. The complete, general formulation can be found in [6].

4.3 Secondary Power Savings

To estimate the impact that isolation of a module c_i has on its fanout candidates in $\mathcal{C}^+_C(c_i)$, let us first consider an isolation candidate c_i whose output is directly connected to the input A of a fanout candidate c_j (Figure 3). By isolating c_i, the following power is saved in c_j whenever c_i is inactive:

$$\Delta P_s(c_i) = \Pr(\overline{f_{c_i}}) \cdot \big(p_j(\mathrm{Tr}_A(c_j), \mathrm{Tr}_B(c_j)) - p_j(0, \mathrm{Tr}_B(c_j))\big), \tag{4}$$

where $\Delta P_s(c_i)$ denotes the secondary power savings obtained by isolating c_i. Eq. (4) models the fact that whenever c_i is inactive the toggle rate at its output—and therefore at input A of c_j—is reduced to zero.

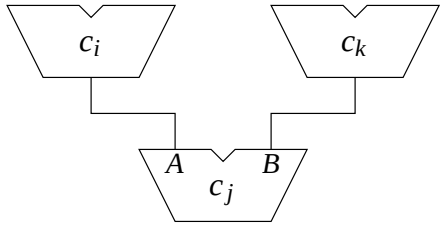

Figure 3: Scenario in which isolation of c_i reduces the toggle rate at the input of its fanout candidate c_j.

However, c_j in turn might have already been isolated, so that whenever c_i *and* c_j are simultaneously performing non-redundant computations, isolation of c_i does not influence power consumption in c_j. We model this fact through a binary decision variable z_j:

$$z_j = \begin{cases} 1 & \text{if } c_j \text{ has been isolated,} \\ 0 & \text{else.} \end{cases} \tag{5}$$

Eq. (4) must hence be refined to

$$\begin{aligned}\Delta P_s(c_i) = \Big[\Pr(\overline{f_{c_i}} f_{c_j}) \cdot \\ \big(p_j(\mathrm{Tr}'_A(c_j), \mathrm{Tr}_B(c_j)) - p_j(0, \mathrm{Tr}_B(c_j))\big)\Big] + \\ \Big[(1 - z_j) \cdot \Pr(\overline{f_{c_i}}\, \overline{f_{c_j}}) \cdot \\ \big(p_j(\mathrm{Tr}'_A(c_j), \mathrm{Tr}_B(c_j)) - p_j(0, \mathrm{Tr}_B(c_j))\big)\Big],\end{aligned}$$

where $\mathrm{Tr}'_A(c_j)$ is the scaled toggle rate according to Eq. (2) if $z_j = 1$, otherwise it is the toggle rate $\mathrm{Tr}_A(c_j)$ measured during simulation. The decision variable z_j in the second term guarantees that power savings induced in c_j by isolating c_i are only taken into account if c_j itself has not been isolated before.

As in the previous section, we can generalize this idea to include a logic network $L_A(c_j)$ that connects c_i to c_j. Let us again consider Figure 2 to compute the secondary savings in a_1 when isolating a_0 assuming that a_1 has *not* been isolated before. We then obtain

$$\Delta P_s(a_0) = \Pr(\overline{AS_{a_0}} AS_{a_1} S_0 \overline{S_1}) \cdot p_{a_1}(\mathrm{Tr}_A(a_1), \mathrm{Tr}_B(a_1)) + \Pr(\overline{AS_{a_0}}\,\overline{AS_{a_1}} S_0 \overline{S_1}) \cdot p_{a_1}(\mathrm{Tr}_A(a_1), \mathrm{Tr}_B(a_1)),$$

since $z_{a_1} = 0$. If a_1 has been isolated before ($z_{a_1} = 1$), the equation simplifies to

$$\Delta P_s(a_0) = \Pr(\overline{AS_{a_0}} AS_{a_1} S_0 \overline{S_1}) \cdot p_{a_1}(\frac{\mathrm{Tr}_A(a_1)}{\Pr(AS_{a_1})}, \mathrm{Tr}_B(a_1)).$$

For a more detailed formulation of the model, the reader is again referred to [6].

5 Automated RTL Operand Isolation

In this section we apply the concepts developed in Sections 3 and 4 in an algorithm for automatically performing operand isolation at the RT level. The algorithm identifies isolation candidates, derives activation functions for them, predicts the impact on timing, power, and area, and isolates the most promising ones.

In Section 5.1 we describe the cost model used to select candidates to be isolated; in Section 5.2, we discuss different isolation implementations, and in Section 5.3 we present the overall iterative isolation algorithm.

5.1 Evaluating Isolation Candidates

Based on the model presented in Section 4, we can estimate the amount of power saved by isolating a module. However, operand isolation also incurs a cost in terms of area, power, and delay due to additional logic introduced into the circuit. When isolating a module, we therefore trade off the *power savings* vs. the *isolation cost*. For analyzing isolation cost, we distinguish two contributing components: the *isolation banks* and the *activation logic*.

Operand isolation affects the timing of an isolation candidate in three ways—the isolation banks increase the delay on the respective paths into which they are inserted, the activation logic creates additional timing paths that merge with the existing paths in the isolation banks, and the activation logic provides increased capacitive loading on every signal used in it. Isolation will therefore decrease the *slack* of the isolated module accordingly. We can estimate the reduction in slack using the timing engine of a synthesis system. Since timing is the most sensitive parameter in many synthesized designs, we will for the time being reject any isolation candidate if its slack drops below a given threshold with isolation.

Both the isolation banks and the activation logic contribute to increased area of the circuit. The *area cost* of the isolation banks is readily given by the number of input bits to isolate. The area cost of the activation logic can be approximated by the literal count of the activation function, which by construction is given in factored form. Let us denote the cost when isolating an isolation candidate c_i by $A(c_i)$. Given the total area estimate A_t of the design, we obtain the relative area increase $\Delta_r A(c_i)$ by

$$\Delta_r A(c_i) = \frac{A(c_i)}{A_t}.$$

Likewise, both the isolation banks and activation logic also incur a *power cost*. The power cost of both these components can readily be estimated since the toggle rates at their input bits (non-isolated) are known after simulating the circuit. Let $P_i(c_i)$ be the power overhead incurred by isolating c_i. The relative change in power $\Delta_r P(c_i)$ using the model from Section 4 is then given by

$$\Delta_r P(c_i) = \frac{\Delta P_p(c_i) + \Delta P_s(c_i) - P_i(c_i)}{P_t},$$

where P_t is the estimated total power consumption of the circuit.

We can combine $\Delta_r A(c_i)$ and $\Delta_r P(c_i)$ in a single cost function. As outlined in Section 4.2 we choose an iterative approach which isolates one isolation candidate in each combinational block at a time. The problem now is to find a set of isolation candidates, at most one from each combinational block, such that a maximum decrease of power consumption is obtained with a minimum increase in area. The cost $h(c_i)$ of isolating an isolation candidate c_i can be stated as follows:

$$h(c_i) = \omega_p \cdot \Delta_r P(c_i) - \omega_a \cdot \Delta_r A(c_i), \qquad (6)$$

where $\omega_p \in [0,1]$ and $\omega_a \in [0,1]$ are weights used to trade off area vs. power: the quotient $\frac{\omega_p}{\omega_a}$ determines the decrease in power consumption that must come with a certain increase in area.

5.2 Isolation Implementations

In addition to using transparent latches for isolating module inputs, we considered implementing isolation using only combinational logic gates, specifically, AND and OR gates. In AND(OR)-based isolation, as opposed to latch-based isolation, the module inputs do not retain their previous values. Instead the AND (OR) gates will force a logic zero (one) at the module inputs during redundant operations. It is clear that AND(OR)-based isolation will result in power savings only if the module is idle for several consecutive clock cycles, a limitation that does not apply to

latch-based isolation. However, AND(OR)-based isolation has several advantages. Since no latches are introduced, the circuit is more amenable for verification, testability, timing, and design-reuse. Additionally, AND/OR gates are less expensive compared to latches in terms of area and power overhead. The effectiveness of these combinational isolation styles is demonstrated in Section 6.

5.3 Putting It All Together

The complete isolation algorithm is given by Algorithm 1. As mentioned in Section 3, we assume that isolation candidates across sequential boundaries do not affect each other. We therefore perform isolation locally in combinational blocks bounded by sequential cells and primary inputs and outputs. The circuit is partitioned into these combinational blocks in line 1, lines 3–11 then identify the isolation candidates in each block, rejecting those candidates which violate the slack conditions discussed above.

Within each combinational block, the algorithm proceeds iteratively: In each iteration (line 13–30), the algorithm estimates the power consumption of the circuit and the signal probability $\Pr(\cdot)$ required to compute the cost functions (line 16). It then isolates the best candidate in each combinational block (line 22). If none of the candidates in a block meet a minimum value $h_{\min}$ of the cost function (line 24), no candidate is isolated. The algorithm terminates when no further improvement can be obtained.

Algorithm 1 Operand isolation on an RT structure

```
1:  {G_1, G_2, ..., G_n} ← partition_RT_structure
2:  {identify isolation candidates and construct auxiliary logic}
3:  for all G_j do
4:    I_j ← isolation_candidates(G_j)
5:    for all c_i ∈ I_j do
6:      estimate_slack_reduction(c_i)
7:      if c_i violates slack conditions then
8:        I_j ← I_j \ {c_i}
9:      end if
10:   end for
11: end for
12: {main loop}
13: repeat
14:   isolation ← false
15:   {estimate power, compute signal statistics}
16:   estimate_power(G)
17:   for all I_j do
18:     {compute cost function for each isolation candidate}
19:     for all c_i ∈ I_j do
20:       h(c_i) ← ω_p · Δ_r P(c_i) − ω_a · Δ_r A(c_i)
21:     end for
22:     {isolate best candidate}
23:     c ← max_{c_i ∈ I_j} h(c_i)
24:     if h(c) ≥ h_min then
25:       isolate(c)
26:       isolation ← true
27:     end if
28:     I_j ← I_j \ {c}
29:   end for
30: until ¬ isolation
```

6 Results

In the following, we shall summarize the results of applying our model to two industrial benchmark circuits. Both designs were data path blocks extracted from more complex designs.

A special characteristic of the first design (`design1`) was that the activation signal of the isolation candidates in the first combinational stage of the design could be controlled from a primary input. Thus, the relationship between power savings and the statistics of the activation signal could be investigated by applying stimuli with different signal statistics to the circuit. Power estimates were obtained using DesignPower [8].

Table 1 summarizes the results for `design1` for the non-isolated design, the isolated design using different isolation styles (i.e. AND-gates, OR-gates, and LATches for isolation with the appropriate modifications in the construction of the activation function) for a representative set of input stimuli. The table also lists the overhead in area and slack induced by the isolation circuitry. We observed power reductions between 12% and 18%, with area overhead as low as 1.3%.

Since `design1`, as explained previously, allowed controlling the activation signal directly from the testbench, we have done an additional set of experiments over varying signal statistics of the activation signal. To study the effect of signal statistics on power savings, we generated a set of testbenches ranging between low and high static probabilities and toggle rates of the activation signal. Average reduction in power consumption varied between 9% and 13%; overall the power reduction varied between approximately 5% in the worst case and 17% in the best case.

Table 2 summarizes the results for `design2`. Since the statistics of the activation signal could not be controlled from the design's environment, we have again listed power reduction for a typical set of stimuli vs. the overhead in area and slack. In this case, the power reduction was subject to considerably less variation; for all three isolation styles a power reduction of approximately 32% was obtained. This came at an average increase in area of 22%.

In most cases, the worst-case slack shrank considerably for both AND- and OR-based isolation in both designs.

	Power		Area		Slack	
	[mW]	*%reduction*	[μm^2]	*%increase*	[ns]	*%reduction*
non-isolated	124.61	n/a	594,342	n/a	3.14	n/a
AND-isolated	107.46	13.76%	601,866	1.62%	3.18	-1.27%
OR-isolated	102.15	18.02%	601,956	1.28%	3.19	-1.59%
LAT-isolated	109.6	12.04%	637,686	7.29%	2.21	29.62%

Table 1: Power consumption and reduction vs. area and slack for `design1`.

	Power		Area		Slack	
	[mW]	*%reduction*	[μm^2]	*%increase*	[ns]	*%reduction*
non-isolated	16.3155	n/a	157,104	n/a	8.97	n/a
AND-isolated	11.1030	31.95%	190,674	21.37%	5.60	37.57%
OR-isolated	11.0932	32.01%	189,360	20.53%	5.15	42.59%
LAT-isolated	11.1052	31.93%	195,948	24.72%	4.54	49.39%

Table 2: Power consumption and reduction vs. area and slack for `design2`.

This should be expected in the general case as operand isolation adds logic to a circuit and therefore potentially degrades the delay on the critical path. For `design1`, however, AND- and OR-based isolation resulted in circuits with a slightly improved slack—this can be attributed to the fact that additional Boolean optimizations were made possible during logic synthesis by the introduction of AND and OR gates, respectively. It is important to note that both designs could be synthesized to meet their constraints despite the reduction in slack.

It is interesting to note that in both cases combinational operand isolation performed as well as or better than LATCH-based. Obviously, the power overhead induced by the latches offset the gains obtained by eliminating the extra transitions in the first cycle of inactivity when choosing gate-based isolation. This was also confirmed by other benchmarks. From the experimental results, we can conclude that LATCH-based isolation does not offer any benefits over gate-based isolation, which would justify the additional constraints on timing and testability required after insertion of latches.

7 Conclusion

In this paper, we have presented the first comprehensive algorithm to automate operand isolation on RT-level. We have discussed a constructive algorithm to derive a Boolean function for each isolation candidate which exactly models its activation condition. We have presented a model to estimate the power savings which can be obtained by isolating a particular module. The model is used to guide an algorithm which iteratively isolates modules until no further improvement can be obtained. We have also validated that contrary to common understanding, operand isolation can and should be performed using pure *combinational* isolation circuitry rather than latches at no loss in power reduction, and with lower area penalty. Results on selected benchmarks demonstrate the efficiency of the approach, with upto 30% reduction in power dissipation.

References

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design*. IEEE Press, 1990.

[2] A. P. Chandrakasan and R. W. Brodersen. *Low Power Digital CMOS Design*. Kluwer Academic Publishers, 1995.

[3] A. Correale. Overview of the Power Minimization Techniques Employed in the IBM PowerPC 4xx Embedded Controllers. In *Proceedings of the 1995 ACM/IEEE International Symposium on Low Power Design*, pages 75–80, Apr. 1995.

[4] H. Kapadia, L. Benini, and G. D. Micheli. Reducing Switching Activity on Datapath Buses with Control-Signal Gating. *IEEE Journal of Solid-State Circuits*, 34(3):405–414, Mar. 1999.

[5] P. E. Landman. *Low-Power Architectural Design Methodologies*. PhD thesis, College of Engineering, University of California, Berkeley, 1994.

[6] M. Münch. *Synthesis and Optimization of Algorithmic Hardware Descriptions*. PhD thesis, University of Kaiserslautern, 1999.

[7] M. Pedram. Power Minimization in IC Design: Principles and Architectures. *ACM Transactions on Design Automation of Electronic Systems*, 1(1):1–66, Jan. 1996.

[8] Synopsys, Inc., Mountain View, CA. *Synopsys Power Products Reference Manual*, v1997.08 edition, 1997.

[9] V. Tiwari, S. Malik, and P. Ashar. Guarded Evaluation: Pushing Power Management to Logic Synthesis/Design. *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems*, 17(10):1051–1060, Oct. 1998.