Leveraging rule-based designs for automatic power domain partitioning

Abhinav Agarwal & Arvind
CSAIL, Massachusetts Institute of Technology
Cambridge, MA USA 02139
Email: {abhiag, arvind}@mit.edu

***Abstract*—Leakage power reduction through power gating requires considerable design and verification effort. We present a scheme which uses high-level design description to automatically generate a collection of fine-grain power domains and associated control signals. We also describe a method of collecting the dynamic activity characteristics of a domain, viz. *total inactivity* and *frequency of inactive-active transitions*, which are necessary to decide the domain's suitability for power gating. Our automated power-gating technique provides power savings without exacerbating the verification problem because the power domains are correct by construction. We illustrate our technique using two wireless decoder designs.**

I. Introduction

Leakage power reduction is increasingly important in hardware design, especially in implementing wireless applications with long standby times [13]. Also, as technology scaling continues, the leakage to active power ratio increases dramatically [5], [9], making leakage power the dominant factor of the power budget. Power gating, *i.e.,* inserting logic to switch off power to parts of the design, is one way to drastically reduce the leakage power. Introducing power gating in a design involves the following steps:

- Dividing the design into power domains and generating power-gating signals for each domain.
- Determining whether it is useful to power gate a particular domain based on the expected dynamic characteristics such as its periods of activity.
- Insertion of isolation logic and power network layout including the sizing of the power switches, and verification of signal integrity.

Generally, the first two steps are done manually while some tools and techniques exist to automate step 3 [4], [8], [16], [21]. In this paper, we provide a technique to automate the first two steps; our technique is orthogonal to the solutions used in step 3.

Since power gating is usually done manually, it is applied only at a fairly coarse granularity, *i.e.,* large power domains with logic, state and clock networks being switched off for hundreds of cycles. This process requires substantial verification effort to ensure that the functionality is preserved. Such power gated designs rely on a global power controller which requires a significant effort to design and integrate. We propose a technique to automatically power gate the majority of the combinational logic on a per-cycle basis. This technique can be applied independently of whether a global power management scheme exists, and to designs of any size and complexity without additional design and verification effort. Using the greater savings of a fine-grained approach, our technique can result in a significant reduction of the total leakage power dissipation. Power gating can reduce leakage, but it incurs the overhead of the switching power associated with the gating transistors and power domains. It also incurs the area cost of power-gating transistors, isolation logic and power network. Thus, a crucial part of the analysis is to determine which power domains are likely to produce net energy savings. Such analysis, by its very nature, is based on *use scenarios* and needs to be captured empirically during the design process.

The main contributions of this paper are providing 1. a novel technique that uses rule-based design descriptions for automatic partitioning of a digital design into power domains and associated gating signals, and 2. automatic classification of power domains for their suitability for power gating based on their dynamic characteristics. We illustrate our technique using two high-performance wireless decoders. These examples will show that unlike global power controllers, our technique introduces no new logic or state to generate power-gating control signals.

Paper Organization: Section II discusses related work in the area of design partitioning and power gating. In Section III, we describe a general technique for generating power domains and using dynamic activity metrics. In section IV, we describe how the use of rule-based designs eases the partitioning problem and collection of metrics, and our algorithm for the same. In Section V, we introduce two realistic wireless decoder designs and discuss their activity metrics. Section VI discusses the power and performance impact of our technique. Section VII discusses some future extensions to our power-gating technique. Finally, we present our conclusions in Section VIII.

II. Related work

Existing literature on *Operand Isolation* [2], [6] has algorithms for identifying operands that are not needed under dynamic conditions. Such work has targeted reducing unnecessary dynamic transitions in designs towards the goal of cutting dynamic and peak power consumption [17]. Chinnery *et al.* [7] have presented analysis and optimization techniques for gating automation once the list of modules to be power gated and the sleep signals have been specified. Usami *et al.* [20] proposed a fine-grained power-gating scheme that relies on

978-1-4799-1071-7/13/$31.00 © 2013 IEEE

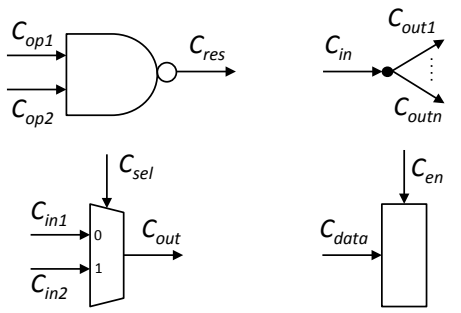


Fig. 1: Nodes of explicit control circuits

already present enable signals of a gated clock design. In contrast, our work does not require any prior analysis of clock gating, and allows gating of logic that might even be feeding registers being clocked in a given cycle. Leinweber *et al.* [12] have described a partitioning algorithm for the design netlist to reduce power consumption. However, starting from the synthesized netlist obscures high-level control signals that can ease the partitioning problem as well as allow utilization of the designer's intent for power reduction. Singh *et al.* [18] have analyzed Term Rewriting Systems for identifying inactive logic for reducing dynamic power. Our work focuses on analysis of similar rule-based designs towards reduction of leakage power, which has increasingly become the dominant factor in recent designs. We also demonstrate how use of dynamic activity statistics is necessary for the selection of appropriate logic domains to power gate.

III. Power domain partitioning

Partitioning a digital design into fine-grain power domains requires identifying control signals that indicate which part of the next-state logic is needed. Two types of control signals are of greatest interest: register write enables and multiplexer selectors. Though such signals are generally quite obvious in high-level designs, they can get obscured in synthesized netlists. Our partitioning technique includes the following steps:

1) A circuit level description of the design given in terms of four elements: Registers, Multiplexers, Forks (for representing fan-outs), and all other logic gates, shown in Figure 1. We call such descriptions explicit control (EC) circuits. Any digital design can be described as an EC circuit.
   Classification of a netlist into EC elements is not unique, for example, a multiplexer is composed of logic gates itself. Rather than identifying such elements in existing netlists, we use a synthesis procedure for rule-based design descriptions where EC circuits arise naturally.
2) A partitioning of the EC circuit into power domains. This step, as described next, is implemented as a graph-coloring task where the set of colors associated with each logic element gives the power gating condition for it.

Our partitioning algorithm is independent of how the EC circuits are generated.

A. Partitioning algorithm as graph-coloring

We describe our power domain partitioning algorithm as coloring a graph consisting of the EC elements as nodes, and connecting wires as links. The definition of the graph colors is as follows:

1) Seed color: Each register (en_i) and mux (sel_j) has a unique seed color.
2) Link color: Each link in the EC circuit has a color which is a Boolean function of the seed colors.
3) Each Mux and logic gate has a color, which is the same as its output link color.
4) Logic elements having the same color constitute a power domain, with the gating condition defined by the color.

Link colors are derived by solving a set of equations which are set up as follows:

- Register i:
$$\begin{aligned} C_{en_i} &= True \\ C_{data_i} &= en_i \end{aligned} \tag{1}$$
- Mux j:
$$\begin{aligned} C_{sel_j} &= C_{out_j} \\ C_{in1_j} &= C_{out_j} \cdot \overline{sel_j} \\ C_{in2_j} &= C_{out_j} \cdot sel_j \end{aligned} \tag{2}$$
- Fork k:
$$C_{in_k} = C_{out1_k} + \cdots + C_{outn_k} \tag{3}$$
- Logic Gate l:
$$\begin{aligned} C_{op1_l} &= C_{res_l} \\ C_{op2_l} &= C_{res_l} \end{aligned} \tag{4}$$
- Constraint due to connectivity: For each link, the sink color is same as the source color.

Given the above set of equations and constraints, it is straightforward to solve for all the link colors in the graph. The solution follows from back-propagation of the seed colors, en_i and sel_j, and gives every link a color which is a boolean expression of seed colors.

B. Example for generation of gating conditions

As an example of this process, consider the segment of a general design shown in Figure 2. Here, each logic block, f_i, consists of a collection of logic gates and forks. The seed colors are the control signals for muxes, and register write enables, called Φ_i, whose generation logic is kept ungated. In this scenario, each logic block has a distinct activity condition consisting of all the control signal values that allow it to propagate to any state element, as shown in Figure 3.

In this manner, power domains and their gating conditions can be generated for an EC circuit. It is important to realize that different power domains dissipate different amounts of leakage power. In fact, when we take into account the energy overhead of switching on the power domain from an inactive state, then it may not make sense to power gate logic that is very active or has short inactive intervals. We next describe how to differentiate between power domains based on their dynamic characteristics.

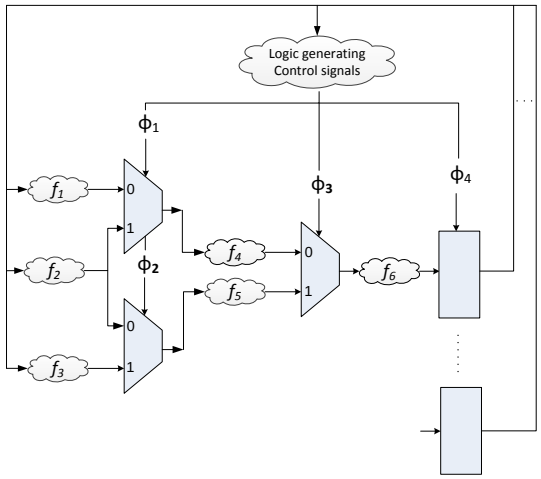


Fig. 2: Selection of logic datapaths

Logic block	Activity Condition
f_1	$\overline{\Phi_1} \cdot \overline{\Phi_3} \cdot \Phi_4$
f_2	$\{\{\Phi_1 \cdot \overline{\Phi_3}\} + \{\overline{\Phi_2} \cdot \Phi_3\}\} \cdot \Phi_4$
f_3	$\Phi_2 \cdot \Phi_3 \cdot \Phi_4$
f_4	$\overline{\Phi_3} \cdot \Phi_4$
f_5	$\Phi_3 \cdot \Phi_4$
f_6	Φ_4

Fig. 3: Activity conditions for logic gates

C. Dynamic Activity Metrics

In order to determine which power domains will provide actual leakage power savings when gated, we need to characterize the dynamic activity of the overall design. There are two ways to use the metrics: first, to see whether the individual power domain should be gated and second, to see whether gating all possible domains of a design saves power. Given a representative testbench or a suite of testbenches, we compute the following dynamic activity metrics:

Metric 1: Total inactivity (N_1) It is defined as the total number of clock cycles in which the logic block under consideration is inactive, *i.e.,* information generated by the block is not used to compute and update any state element.

Metric 2: Frequency of inactive-active transitions (N_2) It is defined as the total number of times the logic block becomes active from an inactive state in the previous cycle.

Use of these metrics depends on the values of two technology-dependent characteristics of the power domain under consideration. Given the desired clock cycle time *t*, let the leakage power saved by gating the logic block be α and the energy cost of switching on the power domain from sleep state be β. Once we have computed these metrics over a realistic testbench, the net energy saved for the i^{th} power domain is given by equation 5. For the complete design, the total energy saved can be computed by summing over all the gated domains as in equation 6.

$$E_i = tN_{1i}\alpha_i - N_{2i}\beta_i \tag{5}$$

$$\sum_i E_i = t\sum_i N_{1i}\alpha_i - \sum_i N_{2i}\beta_i \tag{6}$$

The ideal metrics for power gating a domain would be a high enough number of total inactive cycles (N_1) to compensate for the switching costs of all inactive-active transitions (N_2). Consider a logic block that is inactive for half of the total clock cycles of a test input, but each inactive interval is just one clock cycle, followed by one cycle of activity and so on. In this case, even though its N_1 metric would be quite high, the number of transitions, N_2, would also be very high and would swamp any leakage savings achieved by power gating. In this manner, dynamic metrics allow us to eliminate power domains that are unsuitable for gating due to their expected activity profiles.

For the example in Figure 2, computing these metrics would require information about the dynamic values of the control signals (Φ_i) for realistic testbenches. In general, though this can be done for any EC circuit, doing it at the netlist level is cumbersome and resource intensive as it requires a complete design simulation for very large test inputs and collection of statistical values for each control signal. In the next section, we describe how using rule-based design eases design partitioning and activity metric collection.

IV. Use of rule-based designs

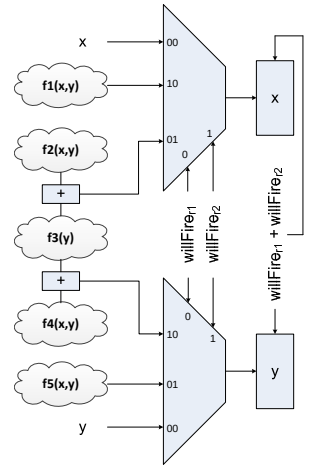
Our technique uses design descriptions that consist of state definitions and $rules$, each of which computes some state updates. Rules are defined as *guarded atomic actions* that can execute in a given cycle if their associated guard condition is true. We use Bluespec System Verilog (BSV) [3] as the source language for rule-based designs. The Bluespec compiler generates a scheduler which selects rules to execute every clock cycle. The scheduler logic ensures that there are no double writes to any register *i.e.,* at maximum only one rule can update a register in a clock cycle. On compilation to Verilog RTL, the compiler generates multiplexers to select the state updates based on these scheduler-generated control signals. In this manner, any EC circuit can be generated from a rule-based design.

As an example, consider the conceptual BSV module shown in Figure 4, consisting of state elements x and y, and two guarded rules $r1$ and $r2$ for state updates. Functions $f1$, $f2$, $f3$, $f4$ and $f5$ used in these rules can be arbitrarily complex combinational functions, which after synthesis can result in large logic blocks. This module description generates the circuit shown in Figure 5(a). The scheduler logic that generates rule firing signals ($willFire_{ri}$) has not been shown as it is not gated. Typically the scheduler logic is much smaller than the rest of the combinational logic of the module. In this example, since both rules update registers x and y, the scheduler ensures that they can never fire in the same cycle.

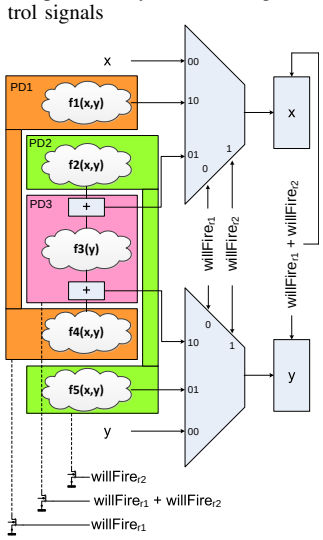
We parse the BSV-generated Verilog to determine the logic blocks that provide data inputs to muxes, and introduce a fine-grained power domain for each input controlled by the corresponding mux control input. Shared logic used to update multiple state elements is gated by the OR of the control signals that select this logic in each mux. The use of BSV rules to automatically generate the power domains ensures correctness by construction. Figure 5(b) shows the possible set of power domains for the example from Figure 4. It consists of three elements: $f1$ and $f4$ in one domain which is turned on

```
module mkExample (InterfaceExample);
   Reg x, y; //state definitions

   rule r1 (p1(x,y));
      x <= f1(x,y);
      y <= f3(y) + f4(x,y);
   endrule

   rule r2 (p2(x,y));
      x <= f2(x,y) + f3(y);
      y <= f5(x,y);
   endrule

   method definitions;
endmodule
```

Fig. 4: Conceptual design of a module in BSV



(a) State updates selected from rule computations by the rule firing control signals



(b) Power domain boundaries and gating conditions

Fig. 5: Generated Hardware

only when $r1$ is selected, $f2$ and $f5$ in another domain which is turned on when $r2$ is selected, and $f3$ in a third domain which is turned on when either rule is selected for execution. The two muxes can also be considered as part of the third domain.

```
Input: Rule-based design and compiled RTL

1. Identify Muxes that select state updates
2. Mux Control inputs are functions of rule firing
   signals Φ
3. Mux Data inputs are potential logic blocks
   with some shared logic
4. Use graph-coloring to select logic functions
   that are controlled by a unique control signal
5. Generate fine-grain UPF power domain description
6. Collect activity metrics to determine leakage
   power savings and switching cost for domains
7. Select power domains that have net power savings

Output: Power Domain definitions for logic blocks
```

Fig. 6: Algorithm to generate and select power domains for rule-based designs

We summarize our algorithm for generating and selecting fine-grained power domains from rule-based designs in Figure 6. We start with an N-rule design, with each rule having a firing condition determining when the rule is active. These conditions, denoted as $\Phi_{1..N}$, are functions of the state within the design. Given these control signals and parsing the generated Verilog code for the mux control inputs, we can associate the firing conditions with the logic blocks used for state updates. This allows us to generate a Unified power format (UPF) [19], [11] specification that provides the place and route tools with the power domain description for each of the logic blocks and their respective gating signals. Finally, we collect activity metrics for the identified logic blocks using rule firing statistics to select appropriate power domains.

Through the use of rule-based digital design description, we can generate hardware that is amenable to the discussed graph-coloring analysis for power domain partitioning as well as collection of dynamic activity metrics for logic blocks. Such designs generate groupings of logic elements and control points, which lead to a natural description in terms of fine-grained power domains. A fundamental issue in power gating is that it should not alter the functionality of the overall design by introducing unintended behavior, such as turning off some needed logic. Grouping logical blocks into distinct power domains, and verifying that they do not introduce changes requires significant verification effort. This is avoided when we power gate rule-based designs.

It is important to note that our technique is not limited to BSV – any design process that generates well-structured RTL code with pre-defined control signals for logic blocks can be used for the method of partitioning that we have outlined. We determine the best candidates for power gating based on a power domain's statistical dynamic activity as discussed earlier in section III-C. This can be done easily for rule-based designs by associating activation of rules with corresponding logic blocks. We demonstrate the collection of such statistics for realistic examples in the next section. This data was collected by automatically instrumenting the BSV designs for rule activity collection and simulating them for various testbenches. This instrumentation does not affect the generated hardware as it is only generated for simulation using compile time macros.

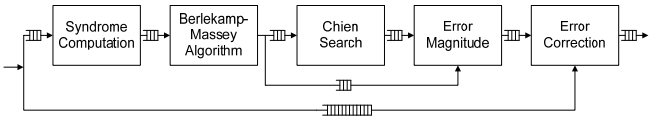


Fig. 7: Architecture of Reed-Solomon Decoder

V. Wireless decoders: Activity metrics

For illustrative purposes, we have chosen two standards-compliant wireless decoders to use for our analysis. We have simulated the designs under realistic traffic patterns to analyze the dynamic activity statistics of their design components.

The first design is a Reed-Solomon decoder [1] which is used in 802.16 transceivers. This design is parameterized by the input block size, and for this study was configured to have 32 bytes of parity in each 255 byte input block. The second design is a Viterbi decoder [15], which is a component in an 802.11 transceiver. Both these designs are high performance implementations that meet performance requirements for respective wireless protocols.

A. Metrics for the Reed-Solomon decoder

Figure 7 shows the architecture of the Reed-Solomon decoder consisting of five modules, each of which performs one step of the decoding algorithm as a finite state machine (FSM). These FSMs are designed such that they might terminate early depending on the dynamic input conditions. We computed the earlier defined activity metrics for each rule in the five modules of the Reed-Solomon decoder under two different input conditions. The data shown in Figure 8(a) corresponds to the maximal activity case which occurs when the number of errors in the input data is equal to the maximum correctable limit determined by the number of parity bytes in the underlying Reed-Solomon encoding. The data was collected over a testbench of 4000 cycles of simulation.

At the other end of the spectrum, Figure 8(b) gives the data for the case where the input data is entirely uncorrupted with no errors. For this input, the first and third modules of the Reed-Solomon pipeline, Syndrome and Chien, still have approximately the same number of inactive cycles (N_1) as the earlier case, while the Error Magnitude computation module is completely inactive as there are no magnitudes to be computed. Berlekamp module also has increased N_1 values, as the rules generating the error magnitude polynomial are inactive.

For the Syndrome module, the entire computation logic is limited to rule *r_in* which is nearly always active, and hence non-viable for gating. For the Berlekamp module, the amount of logic in each *calc_** rule is about the same with no shared logic. Though N_1 is quite high for all the rules, N_2 (number of inactive-active transition) is relatively high for rules *calc_d* and *calc_lambda* as seen in Figure 8(a). Hence, the better candidates for gating are rules *calc_lambda_2* and *calc_lambda_3*. Analysis of the Chien module is similar to that of the Syndrome module. The Error Magnitude module's activity profile depends on whether the input data has errors or not, and hence expectations of the noise characteristics in the use-environment of the module would affect the selection of power domains in this case. Rules *enq_error*, *deq_invalid*, *bypass_int*, and *start_next* can be expected to be good candidates in most cases. The Error Corrector module has a bimodal activity profile where one of two rules fire with a very high activity rate, depending on the presence or absence of errors, making them poor candidates for gating.

Rule	N_1**: No. of Inactive cycles**	N_2**: No. of Inactive-Active transitions**
Syndrome Module		
r_in	24	17
s_out	3984	16
Berlekamp Module		
calc_d	3474	526
calc_lambda	3489	511
calc_lamb_2	3494	506
calc_lamb_3	3494	506
start_new	3984	16
Chien Module		
calc_loc	222	38
start_next	3985	15
Error Magnitude Module		
eval_lambda	261	233
enq_error	3767	233
deq_invalid	3986	14
process_err	714	71
bypass_int	4000	0
start_next	3985	15
Error Corrector Module		
d_no_error	4000	0
d_correct	711	71

(a) Input data with maximal correctable errors

Rule	N_1**: No. of Inactive cycles**	N_2**: No. of Inactive-Active transitions**
Syndrome Module		
r_in	16	16
s_out	3984	16
Berlekamp Module		
calc_d	3472	528
calc_lambda	3488	512
calc_lamb_2	4000	0
calc_lamb_3	4000	0
start_new	3984	16
Chien Module		
calc_loc	32	16
start_next	3984	16
Error Magnitude Module		
eval_lambda	4000	0
enq_error	4000	0
deq_invalid	4000	0
process_err	4000	0
bypass_int	3984	16
start_next	3984	16
Error Corrector Module		
d_no_error	528	16
d_correct	4000	0

(b) Input data with no errors

Fig. 8: Reed-Solomon Activity Metrics

The above analysis was done using a standalone testbench

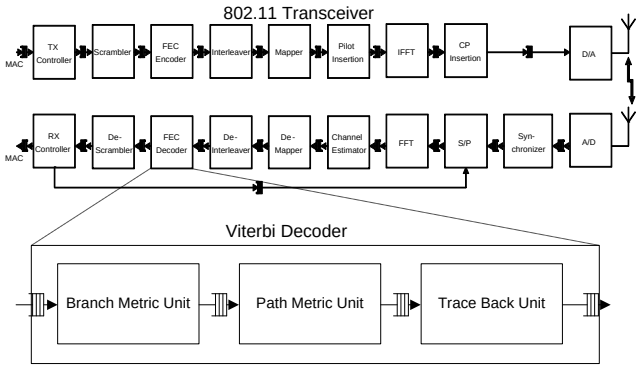


Fig. 9: Architecture of Viterbi Decoder and testbench

for the Reed-Solomon decoder that continuously pumps input data into it. The Reed-Solomon decoder is used in wireless receivers as a Forward Error Correction (FEC) decoder. Depending on the structure of these receiver pipelines, the decoder is not always active. In the next subsection, we use the Viterbi decoder in a complete receiver pipeline to illustrate how external input activity affects the analysis.

B. Metrics for the Viterbi decoder

To obtain the statistical activity data for the Viterbi decoder, we used the AirBlue platform [14] to simulate the complete 802.11 receiver pipeline, as shown in Figure 9. Having a setup of the global architecture in which the decoder itself is a component, provides information on the frequency and *bursty* nature of the input to the decoder. This has an impact on the internal activity metrics - the average activity decreases and the length of inactivity intervals increases. The Viterbi decoder consists of three main modules: Branch Metric Unit (BMU), Path Metric Unit (PMU) and Trace Back Unit (TBU). Each of these modules has all its activity defined in one or two rules. Thus, the data shown in Figures 10(a) and 10(b) is able to capture all the granularity of activity with just four entries (rules *push_zeros* and *put_data* both correspond to BMU). The data was collected over a testbench of 32000 cycles of simulation taken at a steady state. For the case with input errors, we set the SNR at a low value under QAM modulation, such that the Bit Error Rate (BER) of the input data was reasonably high at 0.5%. This setup provides the Viterbi decoder with a significant amount of activity as the wireless receiver pipeline is dominated by the decoding effort. The inactivity metrics for various components of the Viterbi decoder are shown in Figure 10(a). When compared with the large values of inactive cycles (N_1), the infrequency of inactive-active transitions (N_2) indicate that the inactivity intervals for the Viterbi decoder are quite long, much longer than those for the Reed-Solomon decoder, making it an attractive candidate for fine-grained gating even in conditions of low SNR and high activity.

The second testbench environment was set at a high SNR under BPSK modulation, giving an effective BER of zero. As shown in Figure 10(b), under this scenario the Viterbi decoder has a higher number of inactive cycles, as other modules in the pipeline are rate-limiting. The high value of the ratio of N_1 to N_2 emphasizes the long length of their inactivity intervals in this testbench emulating realistic traffic conditions. Based on this data, we can conclude that all of these rules would be excellent candidates for our proposed power-gating scheme.

Rule	**N_1: No. of Inactive cycles**	**N_2: No. of Inactive-Active transitions**
Branch Metric Unit		
put_data	18079	156
push_zeros	30568	38
Path Metric Unit		
pmu_put	15649	156
Trace Back Unit		
tbu_put	15650	156

(a) Input data with BER = 0.5%

Rule	**N_1: No. of Inactive cycles**	**N_2: No. of Inactive-Active transitions**
Branch Metric Unit		
put_data	28376	151
push_zeros	31424	9
Path Metric Unit		
pmu_put	27800	151
Trace Back Unit		
tbu_put	27800	151

(b) Input data with BER = 0%

Fig. 10: Viterbi Activity Metrics

VI. Power and Performance Impact

In this section, we discuss the impact of our technique on the overall power consumption and performance of the designs. As seen in Figure 12, the fraction of total power consumed as leakage in wireless designs is quite significant: 44% in Reed-Solomon decoder and 41% in Viterbi decoder. This fraction increases even further depending on a number of factors such as operating temperature, technology node and relative external activity. This gives us a strong motivation to explore methods for reducing leakage power in such designs.

This data was generated by simulating the extracted place and routed netlists with realistic testbenches. This process also generates the breakdown of the leakage power for each logic element of the design. Next, we need to determine the number of clock cycles for which a typical fine-grained logic block needs to remain inactive for, in order to recoup the energy lost in switching on the power domain and compare it to the dynamic activity metrics obtained from simulation. To determine the switching energy cost per transition and the breakeven threshold of inactivity, we performed circuit-level SPICE simulations of typical logic blocks with appropriately

Design	**Component**	**Power (mW)**
Reed-Solomon decoder	Dynamic	8.624 (56%)
	Leakage	**6.741 (44%)**
	Total	15.365
Viterbi decoder	Dynamic	7.961 (59%)
	Leakage	**5.371 (41%)**
	Total	13.332

Fig. 12: Power consumption breakdown of Reed-Solomon and Viterbi decoders. The synthesis was done using Nangate 45nm library with the operating frequency set to 100MHz. Power consumption data was obtained using simulation of the placed and routed designs.

Property tested		Value
Leakage power of a typical fine-grained and ungated logic block:	a	432.60 nW
Leakage power of the block after gating with a $4/1$ high V_{th} PMOS:	b	0.21 nW
Leakage power saved by power gating the block:	$\alpha = (a - b)$	432.39 nW
Energy lost in turning on the power switch and power domain:	β	20.8 fJ
Breakeven time period for net power savings in a single inactive interval:	β/α	48.1 ns
Breakeven threshold in clock cycles at 100 MHz		5 cycles
Increase in output propagation delay (as measured by 50% output value)		0.31 ns
Increase in output rise time (as measured by 10%-90% delay)		1.50 ns

Fig. 11: Characterization of the breakeven threshold of inactivity interval, and the performance impact for gating a logic block consisting of sixteen 2-input NAND gates with x2 drive strength.

sized power switches using NCSU 45nm library and Nangate Open Cell library. Figure 11 shows a brief summary of the analysis determining the leakage power saved by gating a typical logic block, breakeven time to generate net savings and the impact on output signal propagation due to insertion of power switches and isolation cells.

As seen in the data, by dividing the switching energy cost (β) with the leakage power saved by gating (α), we arrive at the breakeven time period of 48.1 ns. This indicates that the minimum length of inactivity interval required to compensate for the energy lost in switching off the domain is 5 cycles at a clock frequency of 100 MHz. This reflects well on our previous analysis of the decoder designs which had several logic blocks having much longer inactivity intervals, thus providing significant power savings. For each technology library used, designers can similarly determine values of α and β for a typical logic block and use them with the dynamic activity metrics computed for the rule-based design to select appropriate logic blocks to be power gated.

The overall leakage power reduction achieved by our technique can be estimated in the following manner. Leakage power dissipation has four main components: 1. inactive combinational logic used for state updates, 2. the state elements, 3. the buffers of the clock distribution network and 4. the control logic. Our technique targets elimination of leakage power of inactive combinational logic. We categorized the various leakage power components manually by examining the digital power simulation output of the designs under various activity inputs. Based on this analysis, we estimate that 35% of the total leakage power is due to the first component. After accommodating the switching costs of power gating the domains, we can save about 90% of their leakage power consumption. Given that the decoders had up to 44% power as leakage (Figure 12), we estimate that our technique can save up to 14% ($= 0.44 \times 0.9 \times 0.35$) of the total power consumption without any power-saving design burden on the user.

For our gating methodology, the power domains need to be turned on or off within a clock cycle. This places a requirement on the clock cycle time to be long enough to accommodate the time required to turn on a power domain and complete logic computation. For the fine-grained power domains under consideration in the two wireless decoders, the impact of gating is seen as an increase in propagation delay of 0.31 ns (for the output to reach 50% of max value) and a 1.5 ns increase in output rise time, as shown in Figure 11. After adding this delay to the prior critical path of the decoders, we still had sufficient margin to keep the clock frequency at 100 MHz to allow such gating and maintain the performance requirements of wireless standards compliance.

It is possible that for designs with a high clock frequency requirement, insertion of such gating might necessitate increasing the cycle time. Such performance impact can be mitigated by selective addition or removal of power gating from the overall architecture. Since power gating is defined in a fine-grained manner, it is possible to eliminate gating from the logic on the critical path of the clock. It is also expected that such critical logic would be mostly active and hence would not be a viable candidate for fine-grained power gating anyway.

VII. Extensions to the technique

A. Gating local state

The technique for determining appropriate power domains by collecting dynamic activity information applies to logic as well as state elements. There is a significant fraction of leakage power consumed by state elements like registers. However, the loss of information on switching off state elements, and the difficulty of detecting and specifying exactly when it is functionally correct to do so, is a barrier to this technique. Verifying that all future readers of a state have obtained the required information and the value stored in the state element will not be needed by any block internal or external to the module is a complex problem. However, it could be possible to turn off some local state elements inside a module. For example, consider a divider implementation that takes multiple cycles to complete an operation. Any state that is used in this implementation to store running accumulator values or pipeline states, is strictly local to the divide operation and can be turned off when the operation is completed. One way to implement this is to use multi-cycle rule expression, an enhancement to BSV [10], which generates state elements that are only needed when the operation is triggered. Another way could be by compiler analysis of state within a module to determine what state is localized for operations and is not read externally. Efforts in this direction are underway.

B. Allowing turn on over multiple cycles

Currently, our technique requires the power domain to turn on within a clock cycle, thus limiting the maximum clock frequency of the design. One way to overcome this limitation is to allow domains to turn on over multiple cycles. To prevent change in functionality, this requires change in rule scheduling to stall rules that read state updates from a rule being activated over multiple cycles. Development of a modified compilation scheme that generates such scheduling logic is one of the future extensions of our work.

C. Extending to module level gating

Power gating designs at the module-level granularity has some advantages from an implementation viewpoint. Each BSV module can be synthesized as a separate Verilog module and can be easily expressed as an independent power domain. To gate such a domain, we require information about its logic activity as well as state usage. We identify the following conditions for turning on and off a power domain that consists of a single module with *methods* for interfacing with the external environment. To turn the power domain on from an off state, any one of the IO method *enable* signals should be high. To turn the power domain off from an on state, all of the following need to be true: all method *enable* signals should be low, all rule *willFire* signals should be low and none of the internal state elements should be read by any external module.

Before switching off an entire module, we have to ensure that there is no loss of state that is yet to be read by an external method. One naive way of verifying this would be to check the $ready$ signals of output methods. For example, if a module has a FIFO in which the output is stored for an external module to read from, after the computation is completed all rules can stop firing while the output methods become ready. However, this is not a universal fact, and there are instances where module registers need to be read externally and these might be always ready. We propose the use of a *state done* signal that can be combined optionally with one or more methods or it can be an independent signal designed by the user. Here, the burden of ensuring that there is no pending state that can be lost by power gating lies with the designer, making the correctness issue moot. In this scenario, some of the gating logic, namely the rule *willFire* signal generation, is contained within the power domain being gated. However, this logic is only needed for turning off the domain and hence it will be valid when used.

VIII. Conclusion

We have proposed a technique to automate: 1) the partitioning of a high-level design into independent fine-grained power domains with associated control signals, and 2) the collection of dynamic activity information for selecting viable domains from the point of view of leakage savings. For this purpose, we use two technology-independent metrics: *total inactivity* and *frequency of inactive-active transitions*. These activity metrics can be collected automatically during the normal test phase of the design flow. The power-gating control signals are correct-by-construction and do not involve any hardware overhead because they are generated using high-level information already present in rule-based designs. Using two wireless decoder designs, Reed-Solomon and Viterbi, we showed that the use of our technique can reduce the leakage power consumption significantly. The most important aspect of our technique is that it can be used in conjunction with a global power management scheme, accruing additional power savings. Even when a global power domain is turned on, the use of our technique allows switching off parts of it automatically.

Acknowledgement

This work was supported by funding from Qualcomm Inc. and DARPA (under program contract HR0011-13-2-0005).

References

[1] A. Agarwal, M. C. Ng, and Arvind. A Comparative Evaluation of High-Level Hardware Synthesis Using Reed-Solomon Decoder. *Embedded Systems Letters, IEEE*, 2(3):72 –76, 2010.

[2] N. Banerjee, A. Raychowdhury, K. Roy, S. Bhunia, and H. Mahmoodi. Novel low-overhead operand isolation techniques for low-power datapath synthesis. *IEEE Trans. Very Large Scale Integr. Syst.*, 14(9):1034–1039, Sept. 2006.

[3] Bluespec, Inc., Waltham, MA. *Bluespec SystemVerilog Reference Guide*, November 2009.

[4] B. H. Calhoun, F. A. Honore, and A. Chandrakasan. Design methodology for fine-grained leakage control in MTCMOS. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 104–109, New York, NY, USA, 2003. ACM.

[5] B. Chatterjee, M. Sachdev, S. Hsu, R. Krishnamurthy, and S. Borkar. Effectiveness and scaling trends of leakage control techniques for sub-130nm CMOS technologies. In *Proceedings of the 2003 international symposium on Low power electronics and design*, pages 122–127, New York, NY, USA, 2003. ACM.

[6] A. Chattopadhyay, B. Geukes, D. Kammler, E. M. Witte, O. Schliebusch, H. Ishebabi, R. Leupers, G. Ascheid, and H. Meyr. Automatic ADL-based operand isolation for embedded processors. In *conf. on Design Automation and Test in Europe*, pages 600–605, 2006.

[7] D. Chinnery, K. Keutzer, J. Frenkil, and S. Venkatraman. Power gating design automation. In *Closing the Power Gap Between ASIC and Custom: Tools and Techniques for Low Power Design*, pages 251–280. Springer US, 2007.

[8] D.-S. Chiou, D.-C. Juan, Y.-T. Chen, and S.-C. Chang. Fine-grained sleep transistor sizing algorithm for leakage power minimization. In *Proceedings of the 44th annual Design Automation Conference*, DAC '07, pages 81–86, New York, NY, USA, 2007. ACM.

[9] ITRS Working Group. *International Technology Roadmap for Semiconductors: Design*. 2011.

[10] M. Karczmarek and Arvind. Synthesis from multi-cycle atomic actions as a solution to the timing closure problem. In *Proceedings of Intl. Conference on Computer-Aided Design*, pages 24–31, 2008.

[11] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi. *Low Power Methodology Manual: For System-on-Chip Design*. Springer Publishing Company, Incorporated, 2007.

[12] L. Leinweber and S. Bhunia. Fine-grained supply gating through hypergraph partitioning and shannon decomposition for active power reduction. In *Proceedings of the conference on Design, automation and test in Europe*, DATE '08, pages 373–378, New York, NY, USA, 2008. ACM.

[13] S. G. Narendra and A. Chandrakasan. *Leakage in Nanometer CMOS Technologies (Series on Integrated Circuits and Systems)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005.

[14] M. C. Ng, K. E. Fleming, M. Vutukuru, S. Gross, Arvind, and H. Balakrishnan. Airblue: a system for cross-layer wireless protocol development. In *ANCS*, 2010.

[15] M. C. Ng, M. Vijayaraghavan, G. Raghavan, N. Dave, J. Hicks, and Arvind. From WiFI to WiMAX: Techniques for IP Reuse Across Different OFDM Protocols. In *Proceedings of MEMOCODE'07*, Nice, France, 2007.

[16] Y. Shin, J. Seomun, K.-M. Choi, and T. Sakurai. Power gating: Circuits, design methodologies, and best practice for standard-cell vlsi designs. *ACM Trans. Des. Autom. Electron. Syst.*, 15:28:1–28:37, October 2010.

[17] G. Singh and S. K. Shukla. Low-power hardware synthesis from trs-based specifications. In *MEMOCODE*, pages 49–58, 2006.

[18] G. Singh and S. K. Shukla. *Low Power Hardware Synthesis from Concurrent Action-Oriented Specifications*. Springer, 2010.

[19] P. Stanley-Marbell. What is IEEE P1801 (Unified Power Format)? *SIGDA Newsl.*, 37(19):1:1–1:1, Oct. 2007.

[20] K. Usami and N. Ohkubo. A design approach for fine-grained run-time power gating using locally extracted sleep signals. In *24th International Conference on Computer Design*, 2006.

[21] T. Xu, P. Li, and B. Yan. Decoupling for power gating: Sources of power noise and design strategies. In *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*, pages 1002–1007. IEEE, 2011.