Published: 28 September 2025

Citation in BibTeX format

MEMOCODE '25: International Symposium on Formal Methods and Models for System Design
*September 28 - October 3, 2025*
*Taipei, Taiwan*

**Conference Sponsors:**
SIGDA
SIGBED

RESEARCH-ARTICLE

# Automated Power Domain Insertion and Control in Dataflow Circuits

**MARTHA BARKER**, Emory University, Atlanta, GA, United States

**MARK SANTOLUCITO**, Barnard College, New York, NY, United States

**STEPHEN A. EDWARDS**, Columbia University, New York, NY, United States

**MARTHA A. KIM**, Columbia University, New York, NY, United States

# Automated Power Domain Insertion and Control in Dataflow Circuits

Martha Barker
Emory University
Atlanta, GA, USA
martha.barker@emory.edu

Mark Santolucito
Barnard College, Columbia
New York, NY, USA
msantolu@barnard.edu

Stephen A. Edwards
Columbia University
New York, NY, USA
sedwards@cs.columbia.edu

Martha A. Kim
Columbia University
New York City, NY, USA
martha@cs.columbia.edu

## ABSTRACT

Energy efficiency is a key concern in circuit design. With the end of Dennard scaling, voltages no longer scale with transistor size, making full-capacity operation unsustainable due to heat and power limits. Power gating offers a solution, but controlling independent power domains is challenging. Large domains are manageable but inefficient; small domains are efficient but require intricate control.

This paper introduces algorithms, integrated into a high-level synthesis tool, to automate fine-grained power domain partitioning and control. From a dataflow representation, the tool classifies actors into always-on and sleepable domains and generates wake signals. Experiments produced circuits with 48 to 541 power domains, many more than manually feasible, while ensuring correctness despite wake delays. Domains were asleep 60–73%; coordinating wake/sleep cycles for interdependent actors could increase runtime.

## KEYWORDS

power gating; computer-aided design; CIRCT; dataflow

**ACM Reference Format:**
Martha Barker, Mark Santolucito, Stephen A. Edwards, and Martha A. Kim. 2025. Automated Power Domain Insertion and Control in Dataflow Circuits. In *Proceedings of ACM Conference (MEMOCODE '25).* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3742875.3754682

## 1 INTRODUCTION

Leakage power, consumed by inactive transistors, has become the dominant source of power consumption as transistors shrink [17]. Although more transistors now fit in the same area, not all can be powered simultaneously due to heat and battery constraints. Power gating, which turns off idle chip regions, can reduce leakage because only about 10% of logic gates are active at any time [12, 15], allowing up to 90% of leakage power to be saved in theory.

These theoretical savings are rarely achieved in practice, as they require small, fast power gates and the identification of fine-grained power domains. Recent work has produced power gates with single-cycle wake delays [5]; we focus on identifying and controlling many small domains without causing deadlock from sleeping logic.

We present algorithms that automatically insert and control fine-grained power domains, implemented in the high-level synthesis (HLS) tool CIRCT [6]. CIRCT is an open-source hardware compiler that adopts the LLVM/MLIR approach, providing reusable infrastructure for defining intermediate representations called dialects. One such dialect, Handshake, models the circuit in dataflow form.

Dataflow circuits consist of functional units, or actors, that communicate via latency-insensitive handshakes. We use this representation to automatically define power domains and insert control logic that dynamically wakes and sleeps actors at runtime. By exploiting dataflow properties, we identify safe states for power removal and restoration without expensive state retention. We also detect idle and active periods precisely, without software control or prediction. The resulting circuit is deadlock-free and functionally equivalent to the original, regardless of power gate latency.

We lower C code to MLIR [18] using Polygeist [21], then apply existing passes to reach the Handshake dialect. Our transforms operate within Handshake or during the Handshake-to-HW lowering pass, which converts Handshake to the 'hw', 'comb', and 'seq' dialects that represent digital logic. From there, the flow follows the standard path to generate RTL.

This paper enables automated fine-grained power gating with an algorithm to identify a power domain for each dataflow actor; an algorithm to determine safe states for sleeping and reawakening without altering the actor's outputs; an algorithm to synthesize control logic that preserves token sequences regardless of sleep or wake latency; and a fully automated flow, implemented as CIRCT passes, to power gate entire circuits.

## 2 POWER GATING DATAFLOW CIRCUITS

Dataflow representations are increasingly used in HLS to accelerate programs with complex control flow [9, 16, 19]. A dataflow circuit consists of functional units, or *actors*, connected by channels that transfer data as tokens using a latency-insensitive handshake protocol. Each channel includes a *data* signal for the payload, a *valid* signal indicating that data is available, and a *ready* signal in the reverse direction indicating that the receiver is ready. A token transfers only when both *valid* and *ready* are true. CIRCT provides a library of actor implementations consistent with those in [10], with support for custom actors as well.

Analyzing *ready*/*valid* signals and dataflow firing rules allows precise tracking of circuit activity. It also reveals which components control token flow and must remain active to prevent deadlock. Finally, the latency-insensitive handshake guarantees correct operation regardless of actor latency-critical for power gating, where wake delays may impact performance but not correctness.

We start with an ungated dataflow circuit for each actor, treating each independently. We separate data circuitry that can be powered down from control circuitry that stays powered, identify states where power cycling data circuitry won't affect outputs, synthesize a circuit to signal when data circuitry must be awake, and adjust *valid* and *ready* signals to handle wake delays. Finally, we reassemble power-gated actors following the original network topology.

Our approach is local to each actor, enabling power gating of the entire circuit by applying the method independently to each actor.

*Partitioning into Control and Data Planes* First we divide the gates and registers of each actor's circuit into an always-on control plane and a data plane that we will eventually power gate. We define the control plane as gates that drive an actor's outbound *valid* and *ready* signals directly or indirectly:

*Definition 2.1 (Control and Data Planes).* A module $m$ is part of the *Control Plane* if it drives *Valid* or *Ready* output wires, or if it drives another Control Plane module. All other modules are part of the *Data Plane*.

*Defining Idle States* To save power, we want to put the data plane to sleep frequently without altering circuit behavior. Since registers wake with reset values [25], the data plane can safely sleep and reawaken if it is in the reset state. We aim to identify other data plane states where discarding register values and reawakening in reset does not affect future valid tokens. We call these Idle States.

*Definition 2.2 (Idle State).* Let $D$ be the set of registers in the Data Plane; $Reset(r)$ be the reset value of register $r$; $Value(r)$ be the current value of register $r$; and $X$ represent a "don't care" value that could be either 0 or 1. An actor is in an *Idle State* if and only if $\forall r \in D, Value(r) = Reset(r)$ or $Value(r) = X$.

This definition implies that any idle state is observationally equivalent (a hyperproperty [7] relating two traces of a system) to the reset state of the data plane. For a circuit whose data plane is in an idle or reset state, and the control plane is unchanged, simulating the circuit will produce the same sequence of output tokens.

Lemma 2.3 (Observational Equivalence for Idle States). *For any idle state $S$ or reset state $S_{reset}$, an actor $A$ starting from $S$ is observationally equivalent to the same actor $A$ starting from $S_{reset}$ with respect to the sequence of valid output tokens, modulo potential differences in latency.*

*Formally, let $\sigma \in I^{\omega}$ be an infinite input sequence over the input domain $I$ to the actor (the valid, ready, and data signals). Let $\tau(A, S, \sigma)$ denote the trace produced by the actor $A$ starting from state $S$ on input $\sigma$, and let $O(\tau)$ denote the observable (to other actors) behavior of a trace (i.e., the sequence of the valid output tokens). Then,*

$$\forall \sigma \in I^{\omega}, O(\tau(A, S, \sigma)) = O(\tau(A, S_{reset}, \sigma)).$$

*Finding Idle States* We now address accurately identifying idle states. This algorithm is sound but not complete: every state it labels idle is truly idle, though some idle states may be missed.

Correctly identifying idle states depends on determining don't-care values in the data plane. When a *valid* input is 0, its *data* input does not affect any valid output token. Our algorithm uses this by injecting don't-care values via three-valued simulation, where signals can be 0, 1, or $X$ (don't-care). During simulation, all *valid* inputs are set to 0, allowing safe assignment of $X$ to their *data* inputs; *ready* inputs are set to 1 to prompt the actor to flush any valid tokens it holds.

We simulate with standard three-valued semantics: e.g., an AND gate outputs 0 if any input is 0, 1 if all inputs are 1, and $X$ otherwise. The simulation ends when it reaches a previously seen state. If it terminates in a self-loop, we check if the terminal state qualifies as idle and declare it the actor's idle state; otherwise, we conservatively conclude the actor has no idle state and do not power gate it.

*Computing the Wake Signal* We synthesize logic to signal when an actor's data plane must be powered on: (1) the actor is not in an idle state, (2) it is about to leave the idle state, or (3) it is producing a valid output token.

The first and third conditions are straightforward: if the actor is not in the idle state, it may hold important state and must stay powered; if it is offering a token, it must retain that value until consumed. The second condition is also justified—leaving the idle state means the data plane will hold data—but is harder to detect.

To detect when the actor may leave the idle state, we eliminate inputs that cannot affect it. We compute the transitive fan-in of each register with a non-$X$ value in the idle state, tracing through combinational logic and other registers. This yields the *relevant* inputs. Inputs outside this set may affect registers marked $X$, but do not change the idle state since those values are don't-cares.

Next, we simulate one step from the idle state under all combinations of relevant single-bit inputs. While potentially exponential, this is practical since the number of inputs is limited by actor ports, not input bit-width. We exclude vector-valued inputs.

We simulate from the idle state over all combinations of relevant single-bit inputs, producing a Boolean truth table for *transition.* We then synthesize this logic as part of the *wake* signal, defined as:

$$wake = \overline{idle} \cdot awake + transition + \sum_{o \in \text{outputs}} o_{valid}$$

Here, *idle* indicates the circuit is in the idle state, *awake* ensures it is not sleeping (see next section), and the third term checks if any output's *valid* is true. Although the transition condition originally included the idle-state check, Boolean simplification allows it to be omitted since it's already covered by the first term.

*Instrumenting the Circuit* With the necessary information, we power gate the actor by creating a new power domain for all data plane components. Its interface includes all wires crossing between the data and control planes. We add a *wake* input to control power-up and an *awake* output to signal when the domain is active. Since *awake* may trail *wake*, the actor responds only when *awake* is asserted. We drive *wake* with the synthesized logic and gate all *valid* and *ready* outputs with *awake* to block token transfers until the data plane is powered.

Theorem 2.4 (Power Gating Observational Equivalence). *For an actor $A$, and the power gated version of this actor $A'$, the observable behavior (from the perspective of other actors) of $A$ is equivalent to $A'$. Formally, let $\sigma \in I^{\omega}$ be an infinite input sequence over the input domain $I$ to the actor (the valid, ready, and data signals). Let $\tau(A, S, \sigma)$ denote the trace produced by the actor $A$ starting from state $S$ on input $\sigma$, and let $O(\tau)$ denote the observable (to other actors)*

| | | LoC | | Types | | Domains/Idle Period | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | Actors | | Doms. | 0 | <10 | <100 | <1k | <10k | >10k |
| **ADI** | *Alt. Dir. Implicit solver* | 53 | 992 | 65 | 541 | 31% | 0 | 41 | 0 | 19 | 9 |
| **Durbin** | *Toeplitz system solver* | 28 | 285 | 51 | 138 | 36 | 1 | 38 | 0 | 20 | 5 |
| **FDTD 2D** | *2D finite diff. time dom.* | 24 | 646 | 61 | 352 | 38 | 0 | 32 | 0 | 23 | 7 |
| **FloydWarsh** | *Dyn. prog. shortest path* | 13 | 165 | 41 | 76 | 32 | 3 | 36 | 9 | 11 | 11 |
| **Heat 3D** | *3D heat equation* | 30 | 619 | 53 | 335 | 28 | 0 | 45 | 13 | 0 | 14 |
| **Jacobi 1D** | *1D Jacobi stencil* | 16 | 183 | 40 | 90 | 32 | 0 | 49 | 0 | 16 | 3 |
| **Jacobi 2D** | *2D Jacobi stencil* | 16 | 359 | 45 | 186 | 30 | 0 | 44 | 0 | 19 | 7 |
| **LU** | *LU decomposition* | 18 | 254 | 41 | 119 | 30 | 0 | 37 | 0 | 26 | 7 |
| **LU Decomp** | *LU decomp.+fw. subst.* | 37 | 513 | 56 | 252 | 36 | 0 | 15 | 2 | 28 | 19 |
| **MM2** | *Two matrix mults.* | 26 | 384 | 54 | 198 | 36 | 0 | 28 | 0 | 21 | 15 |
| **Nussinov** | *Dyn. prog./seq. align.* | 24 | 598 | 74 | 252 | 25 | 1 | 15 | 0 | 33 | 25 |
| **Seidel 2D** | *2D Seidel stencil* | 12 | 248 | 49 | 120 | 27 | 39 | 10 | 0 | 13 | 11 |
| **TriSolve** | *Triangular solver* | 14 | 117 | 41 | 48 | 27 | 33 | 2 | 0 | 35 | 2 |

**Table 1: Thirteen test circuits: each 12 to 53 lines of C code, yielding 117 to 992 actors (41 to 74 kinds per circuit) split into 48 to 541 power domains. Rightmost columns list percentage of domains per average idle period, typically 10-100.**

*behavior of a trace (i.e., the sequence of the valid output tokens). Then,*

$$\forall S \in \mathcal{S}, \sigma \in I^{\omega}, O(\tau(A, S, \sigma)) = O(\tau(A', S, \sigma)).$$

Proof. We show that the observable behavior of $A$ is equivalent to $A'$ by showing that the two actors produce the same sequence of valid output tokens. Roughly, we aim to show in this proof that we make the correct choice in *what* to power gate (idle states), and *when* to power gate (wake signal). First, we observe that $A'$ differs from $A$ in three ways: (1) partitioning into control and data planes, (2) power gating of the data plane, and (3) gating of valid/ready signals with the wake signal.

From Definition 2.1, the control plane contains all components that drive valid and ready signals. These components are always on, so only data plane power could cause any differences in tokens.

The data plane is only powered down when the actor is in an idle state (Definition 2.2). By Lemma 2.3, any actor starting from an idle state is observationally equivalent to starting from the reset state with respect to output tokens. Since the data plane powers up into its reset state, powering off the data plane when in an idle state does not affect the output token sequence. Because our idle state identification is sound (but not complete), we know that any state we identify as idle with our algorithm is indeed an idle state. Thus, know that we never incorrectly power gate.

From above, the wake signal logic ensures the data plane is powered on in three cases: (1) when not in an idle state, (2) when about to transition from idle, or (3) when producing valid outputs. This ensures the actor is awake whenever computation is needed or tokens are being transferred. Finally, the gating of valid and ready signals with the awake signal may delay token transfers during wake-up but doesn't modify or prevent them. Since dataflow handshaking is latency-insensitive, these delays do not change the sequence of tokens, only their timing. Our definition of observational equivalence is at level of actors, and so allows timing differences.

Thus, $A$ and $A'$ are observationally equivalent: always producing the same sequence of valid output tokens on each input stream. □

As established by Theorem 2.4, this algorithm ensures **the result of computation will be equivalent to the always-on case**. We have shown the sequence of output tokens is unchanged by removing and restoring power and the sleepable domain, and that the sleepable domain will eventually wake when needed; tokens being dropped during the handshake is the remaining danger. In dataflow, tokens are transferred when the producing actor offers a valid token and the receiving actor is ready to accept it. For an actor to drop an input token, its outgoing *ready* signal must be high before it is ready to accept the data, i.e., when the data plane is powered down. This is impossible because the algorithm redefines all outgoing *ready* signals as the logical AND of *awake* and the original value, so when the actor is not awake, all *ready* signals will be zero; no tokens will be transferred. For an actor to drop a valid output token, it must be powered down after it has produced a valid output token and before the downstream actor has accepted it. This is impossible: for an actor to offer a valid token, its *valid* output signal will be true so *wake* will be true, and the data domain will not be powered down before the token is transferred.

## 3 EVALUATION

Through examples, we characterize power domains' dynamic activity and evaluate performance impact as a function of wake time.

*Test Circuit Characterization* We evaluate our approach using PolyBench kernels listed in Table 1 [21]. Each kernel is lowered from C to SystemVerilog using CIRCT to generate three circuit variants: (1) a *baseline* with no power gating; (2) a *gated* version with power domains and control from Section 2; and (3) a *merged-gated* version where adjacent unit-rate actors are merged before gating. Each test circuit in Table 1 contains hundreds of actors, with tens to hundreds partitioned into sleepable and always-on domains—far more than could be manually identified and verified.

Table 1 shows the distribution of power domains by average idle period, ranging from 0 (never idle) to over 10k clock cycles. About one third of domains in each circuit never go idle during simulation. This matches the proportion of buffers, which are never powered down due to their limited idle states and their role in holding tokens. Buffers that rarely hold tokens likely aren't needed.

For domains with idle periods, most last 10 cycles or more. Periods between 10 and 1000 cycles offer good opportunities for fine-grained power gating if identified at runtime. Longer periods (1000+ cycles) are also ideal for gating, though likely detectable by other means. Seidel2D and TriSolve are exceptions, with many power domains idling for fewer than 10 cycles. We hypothesize this is due to branch actors producing tokens on alternating outputs, creating short idle and active phases. In Section 3, we show that selectively gating only actors with longer idle periods can reduce overhead.

*Relative Wakefulness* To assess the power and performance impact of fine-grained gating, we simulate both gated and baseline circuits using cycle-accurate RTL simulation (Verilator). In gated circuits, we impose a wake delay—the number of cycles between *wake* = 1 and *awake* = 1—ranging from 0 to 3 cycles.

First, we examine the relative active time of gated and ungated circuits, shown in the Wakefulness data in Table 2 (left). These values report the percentage of power domains powered on in an average cycle. In all cases, no more than 40% are active per cycle, and this remains consistent across wake delays.

This is because once a dataflow actor emits a result, it must stay awake and hold its output token until the receiver is ready. Longer

| | Un-Merged Power Domains | | | | | | | | Merged Power Domains | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Wakefulness | | | | Cycles | | | | Wakefulness | | | | Cycles | | | |
| Wake Delay: | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 | 0 | 1 | 2 | 3 |
| ADI | .34 | .34 | .33 | .33 | 1.00 | 2.29 | 3.61 | 4.95 | .37 | .37 | .37 | .37 | 1.00 | 1.97 | 2.99 | 4.03 |
| Durbin | .40 | .40 | .39 | .39 | 1.00 | 2.12 | 3.39 | 4.65 | .42 | .42 | .41 | .41 | 1.00 | 1.97 | 3.00 | 4.04 |
| FDTD 2D | .40 | .40 | .40 | .40 | 1.00 | 2.15 | 3.44 | 4.70 | .42 | .42 | .42 | .42 | 1.00 | 1.80 | 2.66 | 3.53 |
| FloydWarsh | .39 | .39 | .38 | .38 | 1.00 | 1.92 | 2.93 | 3.93 | .42 | .42 | .41 | .41 | 1.00 | 1.75 | 2.49 | 3.24 |
| Heat 3D | .33 | .34 | .33 | .33 | 1.00 | 2.28 | 3.56 | 4.88 | .39 | .39 | .39 | .39 | 1.00 | 1.77 | 2.63 | 3.52 |
| Jacobi 1D | .39 | .39 | .38 | .38 | 1.00 | 2.06 | 3.13 | 4.24 | .40 | .40 | .39 | .39 | 1.00 | 1.87 | 2.81 | 3.73 |
| Jacobi 2D | .35 | .35 | .34 | .34 | 1.00 | 2.27 | 3.68 | 5.08 | .38 | .38 | .38 | .37 | 1.00 | 1.85 | 2.78 | 3.71 |
| LU | .36 | .36 | .36 | .35 | 1.00 | 1.92 | 2.82 | 3.75 | .39 | .39 | .39 | .38 | 1.00 | 1.73 | 2.45 | 3.19 |
| LU Decomp | .38 | .38 | .38 | .38 | 1.00 | 1.85 | 2.85 | 3.86 | .40 | .41 | .40 | .40 | 1.00 | 1.69 | 2.34 | 3.04 |
| MM2 | .40 | .40 | .40 | .40 | 1.00 | 1.92 | 2.87 | 3.84 | .42 | .42 | .42 | .42 | 1.00 | 1.72 | 2.44 | 3.17 |
| Nussinov | .28 | .28 | .28 | .27 | 1.00 | 2.01 | 3.13 | 4.30 | .33 | .32 | .32 | .32 | 1.00 | 1.87 | 2.78 | 3.73 |
| Seidel 2D | .38 | .37 | .36 | .36 | 1.00 | 2.45 | 4.02 | 5.63 | .40 | .40 | .39 | .39 | 1.00 | 1.82 | 2.68 | 3.54 |
| Seidel 2D* | .45 | .46 | .46 | .46 | 1.00 | 1.17 | 1.35 | 1.53 | .43 | .43 | .42 | .42 | 1.00 | 1.33 | 1.69 | 2.03 |
| TriSolve | .37 | .37 | .36 | .35 | 1.00 | 1.80 | 2.68 | 3.61 | .38 | .38 | .37 | .37 | 1.00 | 1.67 | 2.35 | 3.02 |
| TriSolve* | .44 | .44 | .44 | .44 | 1.00 | 1.00 | 1.01 | 1.02 | .44 | .44 | .44 | .44 | 1.00 | 1.14 | 1.32 | 1.49 |

*Here, we eliminated any power domain with an average idle period under 10 cycles.

**Table 2: Runtime in clock cycles and wakefulness of gated circuit relative to ungated baseline. The measurements on the left are of power gating applied to an unmodified circuit, and on the right after merging unit-rate actors.**

wake delays stretch each active period proportionally, slowing token transfers but preserving token order. As a result, the average number of active domains per cycle remains unchanged.

*Relative Performance* Table 2 shows the the clock cycle runtimes of the power-gated circuits compared to the ungated baseline. The slowdowns are substantial because fine-grained power domains cycle frequently and have short active periods.

Our method identifies many power domains that may not be needed. For example, we removed domains with average idle periods under 10 cycles in the TriSolve and Seidel 2D circuits, raising relative wakefulness from 36% to 44% and 37% to 46%, but cut performance overhead to at most 1.02× and 1.53× the ungated baseline.

Multi-cycle wake delays can be amortized by merging actors into larger power domains that are cycled less often. To test this, we merged adjacent unit-rate actors into a single actor before applying our algorithms. Unit-rate actors have short active periods since they hold no state and stay awake only when new work arrives. They also depend on downstream state-holding actors being awake before accepting inputs. Waking multiple adjacent unit rate actors adds delays since each must fully wake before the next begins.

The results of merging adjacent unit rate actors before power gating appear on the right side of Table 2. Relative wakefulness increases with merging since the entire merged actor must stay awake if any part is active. This rise in activity is balanced by faster overall execution. However, slowdowns remain significant depending on the wake delay. Further merging may help, such as combining actors at divergence and convergence points of disjoint execution paths into single power domains.

*Synthesis Results* We used OpenLane [23] with the SkyWater SKY130 PDK to synthesize and map each test circuit configuration. Instance areas ranged from 73,857 to 842,267μm$^2$. For most gated circuits, area overhead compared to baseline ranged from 4% to 40%, averaging 22%. Two outliers, ADI and Durbin, had overheads of 204% and 222%. Their baseline configurations failed to complete synthesis, suggesting design issues affecting the flow.

## 4 RELATED WORK

Manual power gating is difficult because designers must insert power switches and verify functional correctness and timing for each domain. One automated approach uses clock-gated netlists to partition circuits into power domains [3, 8, 13]. These methods find clock gating conditions from circuit don't-cares, then group clock-gated registers into power domains. Upasani et al. focus on timing for circuits with both clock and power gating, using placement-aware clustering to reduce overhead [27]. Udupi et al. use formal verification to identify temporary unobservabilities for clock and power gating [26]. Usami and Yoshioka reduce leakage by detecting when state transitions do not occur [28]. All require pre-existing clock gating; our method derives this information directly from the dataflow circuit structure without that step.

Another approach to automating clock gating exploits finite state machine architecture. Agarwal, Dimopoulos, and Shin et al. use an FSM with data path (FSMD) representation, partitioning the machine into data storage and functional circuits [2, 24]. These FSMDs can be further divided into independent sub-FSMDs that power off when the state transitions to another sub-FSMD. Like our method, they define power domains based on a specific representation, but only one power domain is active at a time.

Leakage power is also a concern in reconfigurable architectures, which can be power gated at the LUT or tile level. Ishihara et al. and Bsoul and Wolton designed FPGAs with fine-grained power gating, controlling power domains at runtime via the routing fabric [4, 14]. Miniskar et al. propose a programmer-directed scheme to turn off unused tiles in CGRA architectures, while Nayak et al. reduce area overhead for boundary protection in power-gated CGRAs [20, 22]. These methods target specific devices; our approach is more general.

Like us, other researchers use high-level circuit information to guide power gating. Fanni et al. leverage the dataflow output of the MPEG Reconfigurable Video Encoding standard to map multiple dataflow networks onto a power-gated CGRA [11]. They identify common logic regions that can be powered down when none of the mapped dataflow networks are active. Our work also uses a dataflow representation but partitions a single network and generates control signals from its dynamic activity. Agarwal and Arvind exploit Bluespec's rules-based design to create fine-grained power domains with control signals [1]. Each rule's firing condition determines its activity and thus power gating signals. However, Bluespec's scheduling requires power domains to power on within one clock cycle, limiting clock frequency. Our method supports multi-cycle actors and handles variable wake-up delays.

## 5 CONCLUSION

We present a method to automatically define and control fine-grained power domains in dataflow circuits. Our compile-time algorithm partitions each actor into an always-on control plane and a sleepable data plane. We insert logic to dynamically control the sleep signal and prove that the resulting power-gated circuits are observationally equivalent to the original ones.

## REFERENCES

[1] Abhinav Agarwal and Arvind. 2013. Leveraging rule-based designs for automatic power domain partitioning. In *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 326–333. https://doi.org/10.1109/ICCAD.2013.6691139

[2] Nainesh Agarwal and Nikitas Dimopoulos. 2008. FSMD Partitioning for Low Power Using ILP. In *2008 IEEE Computer Society Annual Symposium on VLSI*. 63–68. https://doi.org/10.1109/ISVLSI.2008.67

[3] Leticia Bolzani, Andrea Calimera, Alberto Macii, Enrico Macii, and Massimo Poncino. 2009. Enabling concurrent clock and power gating in an industrial design flow. In *2009 Design, Automation & Test in Europe Conference & Exhibition*. 334–339. https://doi.org/10.1109/DATE.2009.5090684

[4] Assem A. M. Bsoul and Steven J. E. Wilton. 2010. An FPGA architecture supporting dynamically controlled power gating. In *2010 International Conference on Field-Programmable Technology*. 1–8. https://doi.org/10.1109/FPT.2010.5681533

[5] Joao Pedro Cerqueira and Mingoo Seok. 2017. Temporarily Fine-Grained Sleep Technique for Near- and Subthreshold Parallel Architectures. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 25, 1 (2017), 189–197. https://doi.org/10.1109/TVLSI.2016.2576280

[6] CIRCT. [n. d.]. *Circuit IR Compilers and Tools*. https://circt.llvm.org

[7] Michael R Clarkson and Fred B Schneider. 2010. Hyperproperties. *Journal of Computer Security* 18, 6 (2010), 1157–1210.

[8] Jason Cong, Bin Liu, Rupak Majumdar, and Zhiru Zhang. 2010. Behavior-Level Observability Analysis for Operation Gating in Low-Power Behavioral Synthesis. *ACM Trans. Des. Autom. Electron. Syst.* 16, 1, Article 4 (nov 2010), 29 pages. https://doi.org/10.1145/1870109.1870113

[9] Jordi Cortadella, Mike Kishinevsky, and Bill Grundmann. 2006. Synthesis of synchronous elastic architectures. In *Proceedings of the 43rd Annual Design Automation Conference* (San Francisco, CA, USA) *(DAC '06)*. Association for Computing Machinery, New York, NY, USA, 657–662. https://doi.org/10.1145/1146909.1147077

[10] Stephen A. Edwards, Richard Townsend, Martha Barker, and Martha A. Kim. 2019. Compositional Dataflow Circuits. *ACM Trans. Embed. Comput. Syst.* 18, 1, Article 5 (Jan. 2019), 27 pages. https://doi.org/10.1145/3274280

[11] Tiziana Fanni, Carlo Sau, Luigi Raffo, and Francesca Palumbo. 2015. Automated Power Gating Methodology for Dataflow-Based Reconfigurable Systems. In *Proceedings of the 12th ACM International Conference on Computing Frontiers* (Ischia, Italy) *(CF '15)*. Association for Computing Machinery, New York, NY, USA, Article 61, 6 pages. https://doi.org/10.1145/2742854.2747285

[12] Bibiche Geuskens and Kenneth Rose. 2012. *Modeling microprocessor performance*. Springer Science & Business Media.

[13] Mototsugu Hamada, Takeshi Kitahara, Naoyuki Kawabe, Hironori Sato, Tsuyoshi Nishikawa, Takayoshi Shimazawa, Takahiro Yamashita, Hiroyuki Hara, and Yukihito Oowaki. 2007. An automated runtime power-gating scheme. In *2007 25th International Conference on Computer Design*. 382–387. https://doi.org/10.1109/ICCD.2007.4601928

[14] Shota Ishihara, Masanori Hariyama, and Michitaka Kameyama. 2011. A Low-Power FPGA Based on Autonomous Fine-Grain Power Gating. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 19, 8 (2011), 1394–1406. https://doi.org/10.1109/TVLSI.2010.2050500

[15] Russ Joseph and Margaret Martonosi. 2001. Run-time power estimation in high performance microprocessors. In *Proceedings of the 2001 international symposium on Low power electronics and design*. 135–140.

[16] Lana Josipović, Radhika Ghosal, and Paolo Ienne. 2018. Dynamically Scheduled High-level Synthesis. In *Proceedings of the 2018 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays* (Monterey, CALIFORNIA, USA) *(FPGA '18)*. Association for Computing Machinery, New York, NY, USA, 127–136. https://doi.org/10.1145/3174243.3174264

[17] N.S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J.S. Hu, M.J. Irwin, M. Kandemir, and V. Narayanan. 2003. Leakage current: Moore's law meets static power. *Computer* 36, 12 (2003), 68–75. https://doi.org/10.1109/MC.2003.1250885

[18] Chris Lattner, Mehdi Amini, Uday Bondhugula, Albert Cohen, Andy Davis, Jacques Pienaar, River Riddle, Tatiana Shpeisman, Nicolas Vasilache, and Oleksandr Zinenko. 2021. MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. In *2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO)*. 2–14. https://doi.org/10.1109/CGO51591.2021.9370308

[19] Rui Li, Lincoln Berkley, Yihang Yang, and Rajit Manohar. 2021. Fluid: An Asynchronous High-level Synthesis Tool for Complex Program Structures. In *2021 27th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC)*. 1–8. https://doi.org/10.1109/ASYNC48570.2021.00009

[20] Narasinga Rao Miniskar, Rahul R Patil, Raj Narayana Gadde, Young-chul Rams Cho, Sukjin Kim, and Shi Hwa Lee. 2016. Intra mode power saving methodology for CGRA-based reconfigurable processor architectures. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*. 714–717. https://doi.org/10.1109/ISCAS.2016.7527340

[21] William S. Moses, Lorenzo Chelini, Ruizhe Zhao, and Oleksandr Zinenko. 2021. Polygeist: Raising C to Polyhedral MLIR. In *Proceedings of the ACM International Conference on Parallel Architectures and Compilation Techniques* (Virtual Event) *(PACT '21)*. Association for Computing Machinery, New York, NY, USA, 12 pages.

[22] Ankita Nayak, Keyi Zhang, Rajsekhar Setaluri, Alex Carsello, Makai Mann, Christopher Torng, Stephen Richardson, Rick Bahr, Pat Hanrahan, Mark Horowitz, and Priyanka Raina. 2023. Improving Energy Efficiency of CGRAs with Low-Overhead Fine-Grained Power Domains. *ACM Trans. Reconfigurable Technol. Syst.* 16, 2, Article 26 (April 2023), 28 pages. https://doi.org/10.1145/3558394

[23] Mohamed Shalan and Tim Edwards. 2020. Building OpenLANE: A 130nm OpenROAD-based Tapeout- Proven Flow : Invited Paper. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. 1–6.

[24] Chi-Hoon Shin, Myeong-Hoon Oh, Sung Nam Kim, and Seong Woon Kim. 2011. Fine-grained power gating of datapath using FSM. In *2011 IEEE 2nd International Conference on Networked Embedded Systems for Enterprise Applications*. 1–5. https://doi.org/10.1109/NESEA.2011.6144936

[25] Youngsoo Shin, Jun Seomun, Kyu-Myung Choi, and Takayasu Sakurai. 2010. Power gating: Circuits, design methodologies, and best practice for standard-cell VLSI designs. *ACM Trans. Des. Autom. Electron. Syst.* 15, 4, Article 28 (Oct. 2010), 37 pages. https://doi.org/10.1145/1835420.1835421

[26] Shrinidhi Udupi, Joakim Urdahl, Dominik Stoffel, and Wolfgang Kunz. 2019. Exploiting Hardware Unobservability for Low-Power Design and Safety Analysis in Formal Verification-Driven Design Flows. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 27, 6 (2019), 1262–1275. https://doi.org/10.1109/TVLSI.2019.2906820

[27] Gaurang Upasani, Andrea Calimera, Alberto Macii, Enrico Macii, and Massimo Poncino. 2010. Reducing Timing Overhead in Simultaneously Clock-Gated and Power-Gated Designs by Placement-Aware Clustering. In *Integrated Circuit and System Design. Power and Timing Modeling, Optimization and Simulation*, José Monteiro and René van Leuken (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 227–236.

[28] K. Usami and H. Yoshioka. 2004. A scheme to reduce active leakage power by detecting state transitions. In *The 2004 47th Midwest Symposium on Circuits and Systems, 2004. MWSCAS '04.*, Vol. 1. I–493. https://doi.org/10.1109/MWSCAS.2004.1354035