Automatic Verification and Identification of Partial Retention Register Sets for Low-Power Designs

Yu-An Shih
Princeton University
Princeton, USA
yashih@princeton.edu

Sharad Malik
Princeton University
Princeton, USA
sharad@princeton.edu

Abstract

Today's low-power designs switch between active and standby modes depending on their activity. While the design is in standby mode, its power supply can be turned off to prevent static power consumption. Retention registers are used in these low-power designs to maintain design state even when power is turned off. However, replacing all normal registers with retention registers results in significant area and power overhead. As a consequence, designers often go through the tedious and error-prone process of identifying a partial set of retention registers required to maintain correct design functionality. Further, checking for the correctness of this partial retention set is limited to simulation-based validation. In this paper, we develop model-checking based techniques to formally verify the correctness of a partial retention set for a low-power design. Further, we propose automatic algorithms to help designers identify compact partial retention sets. These algorithms can leverage designers' knowledge to reduce the overall search space and improve scalability. Experiments on open-source designs demonstrate the efficiency and effectiveness of the proposed algorithms. We also show how utilizing design knowledge can help scale our method up to handling designs with about 400K flip-flops.

Keywords

Low-Power Designs, Partial Retention, Model Checking

ACM Reference Format:
Yu-An Shih and Sharad Malik. 2024. Automatic Verification and Identification of Partial Retention Register Sets for Low-Power Designs. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '24), October 27–31, 2024, New York, NY, USA*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3676536.3676758

1 Introduction

To reduce static power consumption, modern System-on-Chip (SoC) designs utilize Power Gating (PG) techniques, where the power supply can be dynamically disconnected from unused blocks in the system. These low-power blocks (designs) switch between active and standby modes by interacting with the power controller through handshake mechanisms at their interface. During standby mode, the design indicates that it is not performing useful tasks and can be disconnected from the power supply.

ICCAD '24, October 27–31, 2024, New York, NY, USA

ACM ISBN 979-8-4007-1077-3/24/10
https://doi.org/10.1145/3676536.3676758

State Retention [24, 36, 34, 20, 16] techniques are commonly applied to these low-power designs to prevent loss of design state. This approach replaces (normal) registers with specially designed low-leakage retention registers that can retain their values even without power supply (or with a backup power supply with much lower voltage level). The design, preserving its state throughout power-down, can thus maintain correct functional behavior after being powered up.

The main disadvantage of State Retention is its area overhead. A retention register is typically more than 20% larger than a normal register [10]. The existence of retention registers also complicates power control, resulting in further increased wire length and area overhead [17, 23]. Moreover, the gate leakage current of retention registers during power-down also contributes significantly to total leakage current [25].

Fortunately, not all registers in the design need to retain their values during standby mode. For example, if the effect of a non-retained register value never reaches any output ports, the design could still be considered as functionally correct. This approach of identifying a sufficient subset of retention registers is called *partial retention*, as opposed to the otherwise *full retention* approach. The Unified Power Format (UPF) [14], a widely adopted industry standard for low-power electronic systems, allows hardware designers to specify their own retention strategy as part of the power intent for their design. A previous study [22] implemented a physical design flow on a design where 4% of its registers are identified as retention registers, and reported a 16% area saving and 96% static leakage decrease compared to the full retention approach, demonstrating the effectiveness of the partial retention approach.

However, identifying a compact retention register set and verifying the functional correctness of the corresponding partial retention design can be a daunting process. As part of the hardware design flow, engineers in the design team are responsible for manually selecting retention registers based on their understanding of the design. The verification team is then required to create simulation testbenches to verify the correctness of the partial retention design. It often takes several iterations before both teams agree on a compact, though not formally proven to be correct, retention set. It is also common for some industry teams to just stick with the full retention approach to avoid this tedious and error-prone process.

Prior academic works have suggested several automatic approaches to mitigate this problem. For example, Greenberg *et al.* [12, 11] derived formal criteria to determine whether individual registers require retention. Their methods automatically specify an over-approximated retention set for a low-power design, but provide no way to verify a manually identified retention set (when it is different from the automatically identified one). Therefore, these

approaches provide limited help when hardware designers come up with possible retention sets based on their own understanding of the design. Chiang *et al.* [5] proposed the most scalable approach to date, identifying partial retention sets for designs with about 22K flip-flops in their paper. However, the approach is based on analysis of specific test sequences of the design, and the identified retention set is only correct with respect to the analyzed test sequences. Engineers still suffer from the daunting process of creating simulation testbenches, and there is no guarantee that these testbenches cover all corner cases of the design. A detailed description and discussion of these prior works will be provided in Section 6.

In this paper, we propose a formal and automatic approach for partial retention verification and identification. The verified retention set is guaranteed to preserve correct functionality of the design with respect to all possible input sequences. We also demonstrate how design knowledge can be used to enhance the scalability of our approach to handle designs with up to 400K flip-flops. Our contributions are summarized as follows:

- We propose a formal verification framework that allows hardware designers to check the correctness of a given partial retention set for a low-power design.
- Based on the verification framework, we develop algorithms to automatically identify compact partial retention sets for a design. These algorithms can leverage information provided by the designer to aid the identification process.
- We address the lack of benchmarks in this space through developing a benchmark suite that can be used to evaluate the algorithms/tools for the verification and identification of partial retention designs.
- We evaluate our verification and identification algorithms using the proposed benchmark set demonstrating their efficiency and effectiveness.
- We also perform case studies on components, with up to 400K flip-flops, of the NVIDIA Deep Learning Accelerator (NVDLA) [21] to demonstrate how design knowledge can be utilized to enhance the scalability of our approach.

To the best of our knowledge, this is the first open-source framework for partial retention verification and identification. The tools to set up the verification framework and identify partial retention sets, as well as the benchmarks are released at https://github.com/Yu-An-Shih/PartRet.

The rest of the paper is organized as follows. Section 2 provides some background and the problem statement. Section 3 describes our partial retention verification framework. Automatic algorithms to identify compact partial retention sets are proposed in Section 4. In Section 5, we evaluate the proposed algorithms, and demonstrate how our approach can be scaled up to handle industrial designs. Finally, we discuss related work in Section 6, and provide concluding remarks in Section 7.

2 Preliminaries

2.1 Abstract Model for Retention Registers

There exist various implementations of retention registers [24, 36, 34, 20, 16]. In general, a retention register can be viewed as controlled by a pair of SAVE and RESTORE signals. When the SAVE signal is asserted, the register value in the current cycle is saved. When the RESTORE signal is asserted, the register restores the previously saved value. Without loss of generality, we assume all retention registers in the design share the same SAVE and RESTORE signals.

2.2 Low-power Interfaces

Modern low-power designs interact with power controllers through low-power interfaces. For example, Figure 1 shows a handshake sequence of the Q-Channel interface protocol, described in the widely used Arm AMBA Low Power Interface Specification [1]. The QREQn signal is set by the power controller to request transition between running (active) and quiescent (standby) modes. The QACCEPTn signal is used by the design to indicate acceptance of the requests. During standby mode, the design is idle and may be powered down by the power controller.

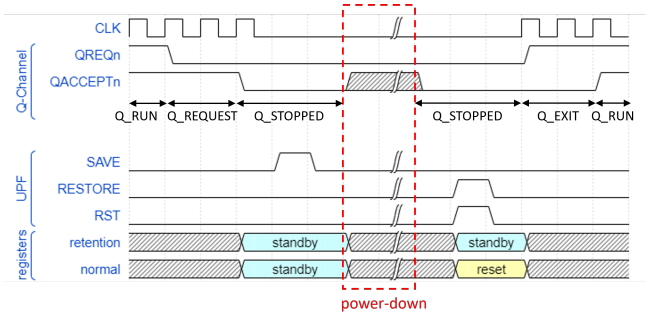


Figure 1: Low-power interface (Q-Channel[1]) and power-down sequences. The low-power interface is part of the design's RTL implementation. All other power-related signals and implementations are specified by the UPF scripts.

Before power-down, the SAVE signal should be asserted for retention registers to save their values. After power-up, the RESTORE signal should be asserted for the values to get restored at some cycle before the design returns active. For all other normal registers, a commonly applied strategy is to reset their values when the RESTORE signal is asserted.

In a typical design flow, the low-power interface is considered as part of the design's *functional intent* and realized in the RTL implementation. All other power-related tasks, including specifying retention registers, setting the SAVE and RESTORE signals, as well as powering down and up the circuit, are the design's *power intent* captured by the UPF scripts.

A low-power design does not interact with the rest of the system when it is powered down. Therefore, when considering the design's functional behavior, the power-down region (as shown in Figure 1) can be safely ignored. Also, the design's RTL implementation, without specifying its power intent, assumes all registers do not change their value during standby mode. We can thus conclude the following:

Observation 1. An RTL design, without specifying its power intent, has the same functional behavior as the full retention setting.

Observation 2. A partial retention design has the same functional behavior as its full retention version, except that each

[1]The Q-Channel handshake sequence is based on Figure 2-2 in the Arm AMBA Low Power Interface Specification [1].

normal register gets reset when the RESTORE signal is asserted.

Note that we will use the Q-Channel Specification as an illustrative example when describing the details of low-power interfaces as it is widely used in the industry. Our proposed methodology and algorithms are generally applicable to other low-power interface protocols.

3 Partial Retention Verification Framework

Figure 2 demonstrates the proposed partial retention verification framework. As stated in Observation 1, the original RTL implementation of a low-power design has the same functional behavior as its full retention version. Then, given a candidate partial retention set, a corresponding partial retention design will be generated. The retention set is valid if and only if the partial retention design is functionally equivalent to the original design under all possible input sequences, including powering down and up the designs.

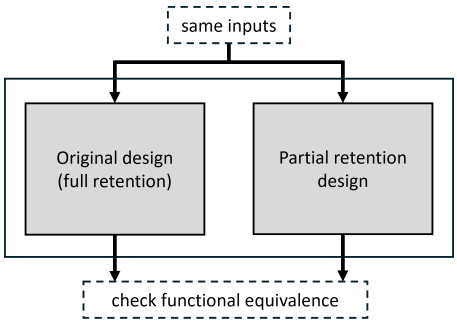


Figure 2: The partial retention verification framework. The partial retention design is generated based on the original design and a partial retention set. The functional correctness of this partial retention design is then verified against the original design.

In this section, we elaborate in detail how to (1) generate a partial retention design based on a candidate retention set and (2) define and verify its functional correctness with respect to the original full retention design.

3.1 Generating Partial Retention Designs

An RTL design can be defined as a tuple $M = \langle S, I, O, S_0, \delta, \omega \rangle$, where

- S is the set of state variables (registers) with space $\mathbb{S}$.
- I is the set of input variables with space $\mathbb{I}$.
- O is the set of output variables with space $\mathbb{O}$.
- $S_0 \in \mathbb{S}$ is the initial (reset) state.
- $\delta : (\mathbb{S} \times \mathbb{I}) \rightarrow \mathbb{S}$ is the state transition function.
- $\omega : (\mathbb{S} \times \mathbb{I}) \rightarrow \mathbb{O}$ is the output function.

For a low-power design, the state transition function δ should satisfy the constraint that no registers change their values during standby mode. According to Observation 1, this corresponds to the full retention setting.

Based on Observation 2, the only modification required to generate a partial retention design is to reset all normal (i.e., non-retention) registers when the user-provided RESTORE input is asserted. Therefore, a partial retention version of the RTL design M can be defined by the tuple $M' = \langle S, I', O, S_0, \delta', \omega \rangle$, where $I' = I \cup \{\texttt{RESTORE}\}$, and all other elements except the state transition function remain unchanged.

The state transition function δ is a combination of individual update functions of each state variable: $\delta = \{\delta_i : (\mathbb{S} \times \mathbb{I}) \rightarrow \mathbb{S}_i \mid \forall s_i \in S\}$, where $\mathbb{S}_i$ is the space of state variable s_i. Accordingly, the registers in the partial retention design are updated as follows:

$$\delta_i'(\mathrm{S}, \mathrm{I}) = \begin{cases} \delta_i(\mathrm{S}, \mathrm{I}), & \forall s_i \in S_R \\ \texttt{RESTORE} \ ? \ S_0(s_i) : \delta_i(\mathrm{S}, \mathrm{I}), & \forall s_i \notin S_R, \end{cases}$$

where S_R is the set of retention registers, and $S_0(s_i)$ returns the reset value of the register s_i.

We develop a tool, based on Yosys [33], to automatically generate partial retention designs. Yosys is an open-source RTL synthesis framework with various features. The Yosys frontend parses and translates Verilog code to its internal data format - the RTL Intermediate Language (RTLIL). Optimization and synthesis algorithms (passes) can then be implemented on the RTLIL data structures to achieve various goals. For our work, we develop a new Yosys pass for partial retention design generation, and integrate it into our verification framework. Given the original (full retention) RTL design M and a retention set S_R, the pass modifies the design to generate the target partial retention design M'.

An advantage of this method is that it does not require any power-related information described by the UPF scripts, except for specifying the retention registers and the RESTORE signal, to generate partial retention designs. Therefore, designers can generate the partial retention designs for verification before more detailed power-related information is provided. Scalability is also greatly improved since there is no need to consider unrelated power implementation details. Moreover, since this method requires only the RTL implementation of the design, any EDA tool that can process Verilog files can be integrated with our framework.

3.2 Functional Correctness of Partial Retention Designs

With a partial retention design M' at hand, the next step is to verify whether it preserves the same functionality as M.

Since the low-power design interacts with other system components through its input and output ports, the functional equivalence between M and M' can be defined as follows: If M and M' receive the same input sequence, they should deliver the same output sequence. This concept is demonstrated in Figure 2.

Let T, T' be the sets of all possible execution traces of M, M' starting from reset. We say a partial retention design preserves *functional correctness* if the following hyperproperty [7] holds:

$$\forall t \in T, t' \in T', (t \approx_I t') \rightarrow (t \approx_O t'), \tag{1}$$

where $t \approx_x t'$ indicates that traces t and t' are indistinguishable with respect to variable set x. A set of retention registers is *complete* if and only if its corresponding partial retention design preserves functional correctness.

This is not the only way to define functional equivalence. For example, since a low-power design only interacts with other system components during active mode, its output values during standby mode does not matter. In this case, we divide output variables O into O_P and O_N. O_P contains the output signals of the low-power interface; O_N contains all other outputs. Given the same input sequence starting from reset, both M and M' should generate the same values for outputs in O_P, as these signals indicate whether the design is in active or standby mode. For all other outputs in O_N, equivalence is required only when the circuit is active.

Again, let T, T' be the sets of all possible execution traces of M, M' starting from reset. The functional correctness of M' can now be defined as:

$$\forall t \in T, t' \in T', (t \approx_I t') \rightarrow ((t \approx_{O_P} t') \wedge (\forall n \in \mathbb{N}, active(t[n]) \rightarrow t[n] =_{O_N} t'[n])), \quad (2)$$

where $t[n] =_x t'[n]$ indicates that states $t[n]$ and $t'[n]$ (n indicates the time step) are indistinguishable with respect to variable set x, and $t \approx_x t'$ indicates that traces t and t' are indistinguishable with respect to variable set x. For the Q-Channel Specification example in Figure 1, $O_P = \{\texttt{QACCEPTn}\}$, and $active(t[n])$ is true if and only if $t[n]$ is in the `Q_RUN` or `Q_REQUEST` regions, i.e., when $\texttt{QACCEPTn} = 1$. Similar conditions can be established for other interface protocols.

The above hyperproperties can be expressed in hardware specification languages such as SystemVerilog Assertions (SVA). Given the design M, the verification framework automatically generates functional equivalence properties in SVA based on Equation 2. Designers can also customize their own properties to better suit the design use case. Once derived, the properties can be verified through standard model checking tools.

This verification approach has similarities to the self-composition technique [2, 27], where two copies of a program are created in the purpose of verifying security properties of the given program. This self-composition technique has also been applied to verify hyperproperties for hardware designs [35].

In summary, given a low-power design M and a candidate retention set S_R, the verification framework automatically generates (1) the partial retention design M' with respect to S_R and (2) the functional equivalence properties with respect to M and M'. The user can then verify the completeness of the candidate retention set through open-source or commercial model checking tools.

4 Automatic Retention Set Identification

With the verification framework described in Section 3, designers can now verify the completeness of their manually-derived partial retention sets. However, the verification framework itself does not provide any information beyond the completeness of retention sets. Designers are still responsible for identifying candidate retention sets for the framework to verify.

In this section, we propose algorithms to help designers identify compact retention sets. Based on the verification framework, these algorithms automatically search for and verify candidate retention sets. These algorithms can also leverage partial information given by the designer to accelerate the search process.

Specifically, the proposed algorithms target the following two scenarios:

(1) Given an incomplete (possibly empty) retention set, find the retention registers required to make it complete.
(2) Given an over-approximated (possibly full) retention set, identify registers that can be removed while ensuring the remaining set is still complete.

4.1 Retention Set Exploration

Algorithm 1 describes our first algorithm for identifying partial retention sets. The user provides the design M, and based on design knowledge, the known retention registers *retention*, known normal registers *normal*, and all other undetermined registers *unknown*. The tool considers registers in the *unknown* set to determine whether they should be put into the *retention* or *normal* sets.

- *Cluster Analysis* (line 1): This function analyzes the registers in *unknown* and creates clusters of "similar" registers based on heuristic algorithms. For example, registers in the same Verilog instance will be grouped into the same cluster; registers whose names differ only in digits or the last suffix will also be grouped into the same cluster. Each cluster represents a guess of a normal register group - when the registers are removed together from a complete retention set, the remaining set may still be complete.
- *Retention Check* (line 5): Each time a candidate group is selected, all *unknown* registers except those in the candidate group are added to the tentative retention set. The algorithm then utilizes the proposed verification framework to generate the corresponding partial retention design and verify its correctness. Based on the result of the retention check, the register sets and similarity groups are updated accordingly.

This algorithm repeatedly selects candidate groups (line 3-4), executes the retention check (line 5), and updates known register sets (line 6-13) until there are no more candidate groups left (i.e., no more *unknown* registers). Theoretically, this algorithm is guaranteed to terminate since the number of candidate groups decreases in each iteration. In practice, the model checker may fail to prove a property before a user-defined time limit is reached.

Algorithm 1 Retention set explorer

Input: Design M, register sets *retention*, *normal*, *unknown*
Output: *retention*, *normal*

```
1:  pool ← Cluster_Analysis(unknown)
2:  while pool ≠ φ do
3:      candidate ← Select_Largest_Group(pool)
4:      ret_target ← retention ∪ (unknown \ candidate)
5:      result ← Ret_Check(M, ret_target)
6:      if result = PASS then
7:          normal ← normal ∪ candidate
8:          pool ← {p \ candidate | ∀ p ∈ pool}
9:      else if result = FAIL and len(candidate) = 1 then
10:         retention ← retention ∪ candidate
11:         pool ← {p \ candidate | ∀ p ∈ pool}
12:     end if
13:     unknown ← unknown \ (retention ∪ normal)
14: end while
```

The retention set explorer (Algorithm 1) addresses both scenarios mentioned at the beginning of this section. Given an incomplete retention set, the user can place the known retention registers in *retention* and all other registers in *unknown*. Given an over-approximated retention set, the user can place the known normal registers in *normal* and all other registers in *unknown*. Therefore, based on their understanding, designers can identify just the obvious retention and normal registers, leaving the difficult parts to the automatic algorithm.

A downside of this algorithm is that it relies heavily on the verification framework's ability to prove completeness of a partial retention set. When the retention check fails, this algorithm does not gain any further information, unless the candidate group contains only one register (Algorithm 1 line 9-11). Generally, it is often more difficult for a model checker to prove correctness of a property than finding a counterexample (CEX) which requires less time and memory. The effectiveness of the cluster analysis algorithm also greatly affects the execution time of this algorithm. In light of these observations, we develop a CEX-guided retention register search algorithm to complement this set exploration method.

4.2 CEX-guided Retention Register Search

The concept of the CEX-guided algorithm is illustrated in Figure 3. To start with, an incomplete (possibly empty) retention set is verified by the retention checker. Since the set is not complete, the model checker will generate a counterexample during retention check. We then implement a simulation-based algorithm (described in Algorithm 2) to find the additional retention registers needed to eliminate this counterexample. The updated retention set is repeatedly sent to the retention checker until no more counterexamples can be generated.

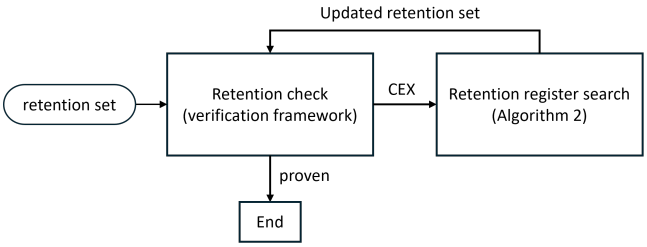


Figure 3: CEX-guided retention register search.

Algorithm 2 describes the details of identifying retention registers to eliminate a given counterexample. The counterexample trace *cex* is a sequence of input, output and register values that demonstrates failure of the target property for an incomplete retention set *retention*. The algorithm finds the additional retention registers needed to eliminate *cex*.

- *CEX analysis* (line 1): A normal register can contribute to functional inequivalence only if its full and partial retention values differ when the RESTORE signal is asserted. All such registers are identified by an analysis procedure and stored in the *candidate* set. A simulation testbench is also generated to check whether a new retention set can eliminate the counterexample.
- *Simulation-based register selection* (lines 2-10): Note that *candidate* is a sufficient but over-approximated register set that can eliminate the given counterexample. We drop registers from *candidate* one by one and simulate the testbench to see whether the counterexample is still eliminated. If not, the register is added to the new retention set.

Since Algorithm 2 always finds a sufficient set of additional retention registers to eliminate the generated counterexample, the CEX-guided search algorithm (Figure 3) is guaranteed to terminate. Again, in practice it may timeout due to excessively long proof time.

Algorithm 2 Retention register search based on a CEX

Input: Incomplete retention set *retention*, counterexample *cex*
Output: Additional set of retention registers *ret_new*

```
1:  candidate, testbench ← Cex_Analysis(cex)
2:  ret_new ← {}
3:  while candidate ≠ ϕ do
4:      reg ← Select_One_Register(candidate)
5:      ret_target ← retention ∪ candidate ∪ ret_new
6:      result ← Simulate(testbench, ret_target)
7:      if result = FAIL then
8:          ret_new ← ret_new ∪ {reg}
9:      end if
10: end while
```

Starting from an incomplete retention set, the algorithm efficiently searches for additional retention registers based on generated counterexamples. Only one proof of correctness is required when the retention set is finally complete. Since model checking tools are more efficient at generating counterexamples than proving correctness of a property, this algorithm usually requires much less execution time compared to the retention set exploration approach. The CEX analysis and simulation-based register selection processes also require little time compared to model checking.

The execution time of this algorithm can be effectively reduced if the designer is able to manually identify some retention registers in advance. Previously known normal registers also helps accelerate Algorithm 2, which looks for missing retention registers in the *unknown* set.

Note that both the cluster analysis in retention set exploration and the simulation-based register selection in CEX-guided retention register search are heuristic algorithms. Therefore, they are not guaranteed to find the minimum retention register set.

4.3 Enhancing Retention Set Exploration with CEX-guided Approach

As described in Section 4.1, the retention set exploration algorithm does not gain any information when the retention check fails, unless the candidate group contains only one register (Algorithm 1 line 9-11). This problem can be solved by integrating the CEX-guided approach proposed in Section 4.2.

Algorithm 3 depicts the enhanced retention set exploration algorithm. When the retention check fails and the candidate group contains more than one register, Algorithm 2 is called to identify the registers required to eliminate the counterexample (line 13-16).

Algorithm 3 Retention set explorer w/ CEX-guided approach

Input: Design M, register sets $retention$, $normal$, $unknown$
Output: $retention$, $normal$

```
1:  pool ← Cluster_Analysis(unknown)
2:  while pool ≠ ϕ do
3:      candidate ← Select_Largest_Group(pool)
4:      ret_target ← retention ∪ (unknown \ candidate)
5:      result ← Ret_Check(M, ret_target)
6:      if result = PASS then
7:          normal ← normal ∪ candidate
8:          pool ← {p \ candidate | ∀ p ∈ pool}
9:      else                                          ▷ result = FAIL
10:         if len(candidate) = 1 then
11:             retention ← retention ∪ candidate
12:             pool ← {p \ candidate | ∀ p ∈ pool}
13:         else
14:             ret_new ← Algorithm_2(retention, cex)
15:             retention ← retention ∪ ret_new
16:             pool ← {p \ ret_new) | ∀ p ∈ pool}
17:         end if
18:     end if
19:     unknown ← unknown \ (retention ∪ normal)
20: end while
```

Enhanced by the CEX-guided approach, at least one register will be added to the retention set for each retention check failure. Since the identified retention registers will never appear in future candidate groups, the cluster analysis algorithm can more easily guess normal register groups correctly in future iterations. Thus, the number of retention check iterations needed can be significantly reduced compared to Algorithm 1.

5 Experimental Results

5.1 Developing a Partial Retention Benchmark Suite

There are only a few open-source RTL designs that interact with power controllers through low-power interfaces. Thus, to evaluate the proposed verification framework and retention set identification algorithms, we selected various open-source designs and extended their interface with low-power interface protocols, allowing them to switch to standby mode during idle periods. We have open-sourced this benchmark suite along with our algorithms.

The benchmark covers a variety of applications, including processors, memory controllers, communication protocols and deep learning accelerators. We obtained three designs from OpenCores: "spi" [26] is a master core of the Serial Peripheral Interface; "mem_ctrl" [28] is a universal memory controller; "wb_dma" [29] provides DMA transfers between two WISHBONE interfaces. PicoRV32 [32] is a popular size-optimized RISC-V CPU core that can be configured to implement the RV32I instruction set. We also perform case studies on two blocks - "cmac" and "cacc" - in the NVIDIA Deep Learning Accelerator (NVDLA) [21].

Among these designs, only the "mem_ctrl" core has its own low-power interface which supports suspending the attached memory devices and the memory controller itself. For all other designs, we extend their input and output interfaces with the Q-Channel protocol [1] described in Section 2.2.

5.2 Evaluating Retention Identification Algorithms

In this section, we evaluate and compare two retention set identification algorithms: (1) The retention set exploration algorithm, enhanced by the CEX-guided approach (i.e., Algorithm 3) and (2) the CEX-guided retention register search algorithm (i.e., Figure 3). Algorithm 3 is subtractive in nature in that it gradually removes registers from $retention \cup unknown$, while Figure 3 is additive in nature in that it gradually adds registers to $retention$.

All experiments were conducted on a 2.4 GHz AMD EPYC 32-core processor with 1TB RAM. We use JasperGold (version 2021.03) [15] as the model checker in our verification framework. For the CEX-guided retention register search algorithm, we use the open-source Icarus Verilog simulator (version 11.0) [31] for the simulation-based optimization step.

The results are shown in Table 1. Each design's name and size are listed in the first two columns. The size of a design is measured in terms of the number of word-level registers and flip-flop bits in the Verilog RTL files. For each algorithm, the number of identified retention registers and the runtime are recorded. The "picorv32" CPU core implements a register file with 32 32-bit registers. 31 of them are architectural state variables and must be retention registers. We include them in the retention set right away before executing the algorithms. For all other designs, the algorithms are executed with no additional design information provided. The runtime is constrained to 24 hours, and "-" indicates a timeout of the algorithm.

Though both algorithms are based on heuristic approaches, they identify the same retention set for most designs. This provides support to the likelihood that these identified retention sets are close to the minimum ones.

For most designs, only a small portion of the registers require retention. This supports our argument that enforcing partial retention can significantly reduce area and power overhead for low-power designs. An exception is the PicoRV32 CPU core, where about 50% of the registers require retention. The reason is that for general-purpose processors, a significant number of registers are used to maintain information across different instructions, and these registers require retention when the design is powered down. Note that the number of identified retention registers in the NVDLA Convolution cores are inferred from their simplified version. The details of simplifying the designs are described in Section 5.3.

In terms of runtime, the CEX-guided search algorithm is more efficient for most designs. To begin with, proving correctness is often more difficult than finding a counterexample for model checking tools. The retention set explorer relies on proving correctness of tentative retention sets to remove groups of normal registers. On the other hand, the CEX-guided approach repeatedly finds missing registers from an incomplete retention set based on finding new counterexamples. Only one proof of correctness is required when the retention set is finally complete. Also, since for most designs the number of retention registers is just a small portion of the total

Table 1: Evaluating the retention register identification algorithms.

Design	# regs/bits	Retention set explorer (Algorithm 3)		CEX-guided search (Figure 3)	
		# retention registers	runtime (hr:min:sec)	# retention registers	runtime (hr:min:sec)
spi	15/231	6 (40.00%)	00:03:04	6 (40.00%)	00:01:22
mem_ctrl	175/1145	5 (2.86%)	01:45:19	4 (2.29%)	00:27:02
picorv32*	184/2328	94 (51.09%)	05:59:13	94 (51.09%)	03:52:20
wb_dma	641/6285	24 (3.74%)	02:25:49	24 (3.74%)	03:17:02
cmac	2781/72932	12 (0.43%)†	-	12 (0.43%)†	-
cmac_simple	18/150	12 (66.67%)	00:01:04	12 (66.67%)	00:00:56
cacc	3723/394532	33 (0.89%)†	-	33 (0.89%)†	-
cacc_simple	42/499	33 (78.57%)	00:10:10	33 (78.57%)	00:09:49

* For this design, 31 registers were identified as retention registers manually before executing the algorithms.
† The number of identified retention registers in this design is inferred from its simplified version.

number of registers, the CEX-guided approach would require fewer iterations of retention check.

The retention set exploration algorithm also has its own advantage. Given a counterexample for an incomplete retention set, the CEX-guided search algorithm looks for missing retention registers in the whole *unknown* set. On the other hand, the retention set explorer, enhanced by the CEX-guided approach (i.e., Algorithm 3 line 13-16), looks for missing retention registers only in the current candidate group, which is a subset of the *unknown* set. Therefore, for designs where the average proof time is relatively short (see "wb_dma" in Table 2), this factor dominates and the retention set explorer could outperform the CEX-guided search algorithm.

We also analyze the effectiveness of cluster analysis for the retention set exploration algorithm. The results are shown in Table 2. The retention and normal registers identified by the retention set explorer are listed in column 2. Without cluster analysis, one can start from a full retention register set, remove each register from the set, and decide whether it stays in the retention set by checking the correctness of the remaining retention set. In this case, the number of CEX iterations would be the number of identified retention registers, while the number of proof iterations would be the number of identified normal registers. For example, the "spi" core would require 6 CEX iterations and 9 proof iterations without cluster analysis.

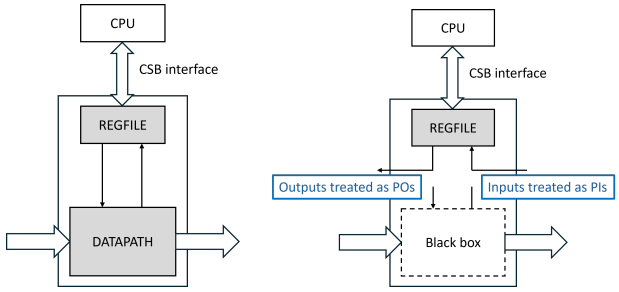
With cluster analysis, our goal is to reduce the number of proof iterations by grouping possible normal registers together. Column 3 shows the number of CEX and proof iterations executed by Algorithm 3 (i.e., with cluster analysis). Comparing to column 2, the cluster analysis successfully saves 33% - 98% of the proof iterations. Also, enhanced by the CEX-guided approach, Algorithm 3 identifies at least one retention register for each CEX iteration. Therefore, the number of CEX iterations is never more than the number of retention registers.

As shown in column 4, the model checker often requires more time to prove a correct retention set than to find a counterexample for an incomplete retention set. Using "mem_ctrl" as an example, the average proof time is 2.39x compared to the average CEX time. This justifies our goal to reduce the number of proof iterations.

5.3 Case Study: NVDLA Convolution Cores

The NVDLA Convolution core [21] consists of several pipeline stages to accelerate the convolution algorithm. We perform a case study on two of the pipeline stages: The Convolution MAC (CMAC) module receives input data and weight from the previous stage, performs multiplication and addition, and outputs the results; The Convolution Accumulator (CACC) module accumulates the partial sums received from CMAC and rounds/saturates the result before sending them to the next stage. As shown in Table 1, our retention set identification algorithms timed out for these designs. In this section, we demonstrate how utilizing design knowledge can help identifying retention sets for designs with such large size.

Figure 4a depicts the high-level view of the CMAC core. The CPU accesses and modifies CMAC's configuration registers in the REGFILE module through a simple configuration space bus (CSB) interface. When configuration completes, the CPU sends an "activate" command to the CMAC core. The DATAPATH then starts executing based on values of the configuration registers and data provided by the previous pipeline stage.



(a) Original design. **(b) DATAPATH black-boxed.**

Figure 4: High-level view of the NVDLA CMAC core.

The low-power interface allows CMAC to switch to standby only when it is not in the middle of an execution phase. Since the execution settings are fully specified by configuration registers in REGFILE, no registers in DATAPATH have to retain their values during standby mode. As shown in Figure 4b, we black-box all modules in DATAPATH to create a simplified version of CMAC. All inputs to REGFILE are treated as primary inputs, and all outputs from REGFILE are treated as primary outputs.

The original NVDLA CMAC core consists of 2781 registers with 72932 bits. After black-boxing the non-retention modules based on design knowledge, the remaining circuit contains only 18 registers and 150 bits. As shown in Table 1, both proposed algorithms can

Table 2: Analyzing the effectiveness of cluster analysis in the retention set explorer.

Design	# retention/normal regs	# CEX/proof iterations	avg CEX/proof time (s)
spi	6 / 9	6 / 4	12.69 / 14.61
mem_ctrl	5 / 170	5 / 11	194.71 / 466.04
picorv32	63*/ 90	52 / 19	279.02 / 339.30
wb_dma	24 / 617	21 / 11	52.85 / 47.56
cmac_simple	12 / 6	10 / 4	0.13 / 0.13
cacc_simple	33 / 9	26 / 4	15.88 / 13.61

* The 31 user-identified retention registers are excluded to more precisely evaluate the effectiveness of cluster analysis.

identify the retention registers in this simplified design within 2 minutes. Only 12, i.e. 0.43% of 2781 registers require retention. The NVDLA CACC core can be simplified and handled in a similar way.

In the above example, we black-box the modules based on the design knowledge that no DATAPATH registers need to be retained. This is not a necessary condition for black-boxing. To clarify, by black-boxing DATAPATH, we focus on identifying retention registers in REGFILE - this does not depend on whether DATAPATH consists of retention registers. Therefore, designers can enforce a divide-and-conquer approach when dealing with large-scale designs - our proposed algorithms can be used to identify retention registers in multiple smaller sub-regions, instead of considering the entire design. In theory, this allows our approach to scale up to arbitrarily large designs. One downside of this approach is that it may lead to over-approximation of the identified retention set, since the inputs to the focused sub-region (e.g. REGFILE) are treated as primary inputs instead of driven by actual design logic. This can be addressed by properly constraining these additional inputs.

So far, this divide-and-conquer process is done manually. We will work on automating the approach in the near future.

6 Related Work

There have been prior works related to identifying and verifying partial retention register sets for low-power designs [8, 12, 11, 5]. Darbari *et al.* [8] proposed a symbolic simulation based approach to help designers implement a partial retention design correctly. Their case study on a simple 32-bit RISC core suggested that the programmer visible states or the architectural states of the CPU be retained, while other micro-architectural states can be left without retention. Greenberg *et al.* [12, 11] formally derived two over-approximated criteria to determine whether a flip-flop (FF) needs to be retained or not. [12] expressed the criteria as Boolean formulas and performed gate-level analysis based on the BDD representation; [11] expressed the criteria in Computation Tree Logic (CTL) and verified them using commercial model checking tools. By checking these criteria, they extract complete sets of retention FFs from designs with up to 3K flip-flops. However, this approach provides no way to verify user-derived partial retention sets. Also, one of the criteria suggests inserting additional combinational logic to the circuit. On the other hand, we focus on a pure verification approach which does not modify the functionality of the circuit. Chiang *et al.* [5] analyzed real test sequences of a design to automatically identify retention registers. The analysis process starts by extracting the initial (standby) state and a fixed-length power-up sequence from a selected test sequence. The hardware design, coupled with this information, was then transformed into formulas in Conjunctive Normal Form (CNF). This allowed them to develop efficient retention identification algorithms based on SAT solvers. The largest design considered in their experiments contains about 22K flip-flop bits. However, the identified partial retention set is not complete unless all possible standby states, with all of their power-up sequences, are considered. They also mentioned that their algorithms are "meant to be used as an analysis tool for designers instead of a verification tool." To the best of our knowledge, our proposed approach is the first open-source and automatic framework that can perform both verification and identification of partial retention designs. Utilizing design knowledge provided by hardware designers, we demonstrate the capability of this approach to handle designs with about 400K bits.

Aside from identifying compact partial retention register sets in RTL, several works focused on minimizing the number of required retention registers as well as other hardware resources during earlier design phases such as high-level synthesis [6, 30]. Some others proposed to use the interesting structure of multi-bit retention flip-flops (MBRFF) [4, 3, 19, 9, 13, 18], as opposed to the so called single-bit retention flip-flops (SBRFF). MBRFF allows saving multiple bits of data in a few consecutive cycles before standby, and restoring by shifting out the data cycle-by-cycle during wake-up. The key concept is that only a selected set of flip-flops should be replaced by MBRFFs, resulting in fewer retention bits in total compared to the SBRFF approach. One downside of MBRFF is the increased transition latency between active and standby modes. These works are orthogonal to ours.

7 Conclusions and Future Work

Today's low-power design teams in the industry often rely on manual or simulation-based approaches to identify and verify partial retention sets. To avoid this tedious and error-prone process, some even choose to stick with the full retention approach despite the significant benefit of partial retention. In this paper, we propose a verification framework to formally check the completeness of a partial retention set. We also provide algorithms that can automatically identify compact retention sets in a low-power design. In terms of scalability, we demonstrate how our approach can be used to handle industrial designs by leveraging design knowledge.

As mentioned in Section 5.3, we will work on automating the divide-and-conquer approach, which helps scale our approach up to handle large-scale designs with increased automation in the near future. Also, so far the implementation of our proposed algorithms is focused on the granularity of word-level registers in Verilog RTL implementations. However, it is possible that only some bits in a register require retention. In the future, we will extend our framework to handle bit-level granularity as well.

References

[1] 2021. AMBA low power interface specification. Retrieved March 16, 2024 from https://developer.arm.com/documentation/ihi0068/c.

[2] Gilles Barthe, Pedro R D'argenio, and Tamara Rezk. 2011. Secure information flow by self-composition. *Mathematical Structures in Computer Science*, 21, 6, 1207–1252.

[3] Yu-Guang Chen, Hui Geng, Kuan-Yu Lai, Yiyu Shi, and Shih-Chieh Chang. 2014. Multibit retention registers for power gated designs: concept, design, and deployment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33, 4, 507–518.

[4] Yu-Guang Chen, Yiyu Shi, Kuan-Yu Lai, Geng Hui, and Shih-Chieh Chang. 2012. Efficient multiple-bit retention register assignment for power gated design: concept and algorithms. In *Proceedings of the International Conference on Computer-Aided Design*, 309–316.

[5] Ting-Wei Chiang, Kai-Hui Chang, Yen-Ting Liu, and Jie-Hong R Jiang. 2015. Scalable sequence-constrained retention register minimization in power gating design. In *Proceedings of the 52nd Annual Design Automation Conference*, 1–6.

[6] Eunjoo Choi, Changsik Shin, Taewhan Kim, and Youngsoo Shin. 2008. Power-gating-aware high-level synthesis. In *Proceedings of the 2008 international symposium on Low Power Electronics & Design*, 39–44.

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

[8] Ashish Darbari, Bashir M Al Hashimi, David Flynn, and John Biggs. 2009. Selective state retention design using symbolic simulation. In *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 1644–1649.

[9] Guo-Gin Fan and Mark Po-Hung Lin. 2017. State retention for power gated design with non-uniform multi-bit retention latches. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 607–614.

[10] David Flynn, Rob Aitken, Alan Gibbons, and Kaijian Shi. 2007. *Low power methodology manual: for system-on-chip design*. Springer Science & Business Media.

[11] Shlomo Greenberg, Joseph Rabinowicz, and Erez Manor. 2014. Selective state retention power gating based on formal verification. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 62, 3, 807–815.

[12] Shlomo Greenberg, Joseph Rabinowicz, Ron Tsechanski, and Eugene Paperno. 2013. Selective state retention power gating based on gate-level analysis. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 61, 4, 1095–1104.

[13] Gyounghwan Hyun and Taewhan Kim. 2019. Allocation of state retention registers boosting practical applicability to power gated circuits. In *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 1–6.

[14] 2019. IEEE standard for design and verification of low-power, energy-aware electronic systems. Retrieved March 24, 2024 from https://standards.ieee.org/ieee/1801/6767/.

[15] 2016. Jasper FPV app. Retrieved March 18, 2024 from https://www.cadence.com/en_US/home/tools/system-design-and-verification/formal-and-static-verification/jasper-gold-verification-platform/formal-property-verification-app.html.

[16] Hailong Jiao and Volkan Kursun. 2010. High-speed and low-leakage mtcmos memory registers. In *2nd Asia Symposium on Quality Electronic Design (ASQED)*. IEEE, 17–22.

[17] Hyung-Ock Kim and Youngsoo Shin. 2007. Semicustom design methodology of power gated circuits for low leakage applications. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 54, 6, 512–516.

[18] Taehwan Kim, Heechun Park, and Taewhan Kim. 2021. Allocation of always-on state retention storage for power gated circuits—steady-state-driven approach. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29, 3, 499–511.

[19] Shu-Hung Lin and Mark Po-Hung Lin. 2014. More effective power-gated circuit optimization with multi-bit retention registers. In *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 213–217.

[20] Hamid Mahmoodi-Meimand and Kaushik Roy. 2004. Data-retention flip-flops for power-down applications. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No. 04CH37512)*. Vol. 2. IEEE, II–677.

[21] NVDLA. Retrieved March 31, 2024 from http://nvdla.org/.

[22] Joseph Rabinowicz and Shlomo Greenberg. 2021. A new physical design flow for a selective state retention based approach. *Journal of Low Power Electronics and Applications*, 11, 3, 35.

[23] Jun Seomun and Youngsoo Shin. 2009. Self-retention of data in power-gated circuits. In *2009 International SoC Design Conference (ISOCC)*. IEEE, 212–215.

[24] Satoshi Shigematsu, Shin'ichiro Mutoh, Yasuyuki Matsuya, Yasuyuki Tanabe, and Junzo Yamada. 1997. A 1-V high-speed MTCMOS circuit scheme for power-down application circuits. *IEEE Journal of Solid-State Circuits*, 32, 6, 861–869.

[25] Youngsoo Shin, Sewan Heo, Hyung-Ock Kim, and Jung Yun Choi. 2007. Supply switching with ground collapse: simultaneous control of subthreshold and gate leakage current in nanometer-scale cmos circuits. *IEEE transactions on very large scale integration (VLSI) systems*, 15, 7, 758–766.

[26] Simon Srot. 2016. Spi controller core. (Apr. 2016). Retrieved April 2, 2024 from https://opencores.org/projects/spi.

[27] Tachio Terauchi and Alex Aiken. 2005. Secure information flow as a safety problem. In *International Static Analysis Symposium*. Springer, 352–367.

[28] Rudolf Usselmann. 2018. Memory controller ip core. (June 2018). Retrieved April 2, 2024 from https://opencores.org/projects/mem_ctrl.

[29] Rudolf Usselmann. 2019. Wishbone dma/bridge ip core. (Mar. 2019). Retrieved April 2, 2024 from https://opencores.org/projects/wb_dma.

[30] Nan Wang, Wei Zhong, Song Chen, Zhiyuan Ma, Xiaofeng Ling, and Yu Zhu. 2018. Power-gating-aware scheduling with effective hardware resources optimization. *Integration*, 61, 167–177.

[31] Stephen Williams. The ICARUS verilog compilation system. Retrieved April 6, 2024 from https://github.com/steveicarus/iverilog.

[32] Claire Wolf. 2019. Picorv32 - a size-optimized risc-v cpu. (Mar. 2019). Retrieved April 2, 2024 from https://github.com/YosysHQ/picorv32/tree/v1.0.

[33] Claire Wolf. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys/.

[34] Hyo-Sig Won, Kyo-Sun Kim, Kwang-Ok Jeong, Ki-Tae Park, Kyu-Myung Choi, and Jeong-Taek Kong. 2003. An MTCMOS design methodology and its application to mobile computing. In *Proceedings of the 2003 international symposium on Low power electronics and design*, 110–115.

[35] Yu Zeng, Bo-Yuan Huang, Hongce Zhang, Aarti Gupta, and Sharad Malik. 2021. Generating architecture-level abstractions from RTL designs for processors and accelerators part i: determining architectural state variables. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[36] Victor Zyuban and Stephen V Kosonocky. 2002. Low power integrated scan-retention mechanism. In *Proceedings of the 2002 international symposium on Low power electronics and design*, 98–102.