Hardware Profiling for SECDA on Edge Platforms

Jude Haris, Rappy Saha, José Cano
School of Computing Science, University of Glasgow, Scotland, UK

***Abstract*—As the need for efficient FPGA-based accelerators for Edge AI increases, the SECDA hardware-software co-design methodology provides guidelines for the efficient, iterative design of specialized hardware accelerators. Fine-grained hardware profiling to identify and alleviate bottlenecks requires additional tooling to profile hardware developed using SECDA during FPGA execution. We present the SECDA Hardware Profiler, a tool that enables fine-grained profiling of hardware designs by integrating hardware monitors into the SystemC accelerator design and by enabling access to these monitors via a new hardware profiling API in the SECDA-Core library. We demonstrate the capabilities of the profiler through a case study of a simple Matrix multiplication accelerator, where we use hardware monitors to track module-specific performance increase across various levels of hardware unrolling. Additionally, we highlight the minimal increase in FPGA resource utilization (around 0.29% LUTs and 0.15% FFs) required to enable the monitoring infrastructure. The SECDA Hardware Profiler will be open-source at: https://github.com/gicLAB/SECDA-Core.**

I. Introduction

As the need for efficient edge AI accelerators increases, improving the design process for these new accelerators is fundamental to developing them with reduced development costs. SECDA [1] is a hardware-software co-design methodology used to efficiently produce new optimized DNN inference accelerators for edge devices using FPGAs. It uses SystemC [2] as an accelerator simulation framework, which allows candidate designs to be iterated upon efficiently. Using SystemC High Level Synthesis (HLS), we can produce a synthesizable design from the same accelerator definition. This approach enables a seamless transition from design to deployment, as the same SystemC code can be used for both simulation and synthesis, reducing the time and effort required to produce efficient hardware accelerators.

Designers can further optimize new designs by using fine-grained hardware execution profiling. While SECDA simplifies the hardware accelerator design process, fine-grained profiling is limited to simulation unless the designer introduces ad hoc hardware profiling. Currently, enabling such profiling requires designers to manually implement and integrate custom hardware monitors into the SystemC design and corresponding software drivers for FPGA execution, which drastically increases the time needed to design a new accelerator. Built-in SECDA hardware profiling during FPGA execution would enable designers to efficiently identify complex bottlenecks.

This paper presents the SECDA Hardware Profiler, which integrates hardware monitors (HMs) into accelerator modules and exposes them through a profiling API in SECDA-Core (SECDA's main hardware-software library). We evaluate the profiler with a case study on a tiled matrix multiplication accelerator with different MAC unrolling factors ($2\times$, $4\times$, $8\times$), revealing clear module-level performance characteristics while introducing minimal area overhead: only 0.17% of total FFs and 0.29% of total LUTs.

Email: jude.haris@glasgow.ac.uk

```
1 void Load();
2 void Compute();
3 void Store();
```

(a) Before Integration

```
1 HWC_Reset;
2 HWC_CTHREAD(Load)
3 HWC_CTHREAD(Compute)
4 HWC_CTHREAD(Store)
```

(b) After Integration

Fig. 1: Modification of accelerator SC_Module definition using SystemC to enable the hardware profiling. Blue indicates modified lines, and green indicates new lines of code.

II. Hardware Profiler

The SECDA Hardware Profiler is enabled by integrating hardware monitors into the SystemC accelerator design and enabling them in software through a new profiling API.

A. Monitors

The hardware monitors (HM) are used to track the cycle counts of the different modules of the accelerator during execution on the FPGA. The hardware monitors are implemented as simple C++ macros that replace SystemC's standard "CTHREAD" definitions. Figure 1 illustrates the code changes required to integrate the hardware monitors into the accelerator's SystemC-based SC_Module definition, where (a) shows the original definition of hardware modules (Load, Compute, Store) and (b) shows the redefinition with new hardware profiler macros.

Each monitor is associated with a specific module of the accelerator, and contains a single input port 'target_state', two output ports 'cycle_count' and 'current_state', and an internal counter variable 'counter'. The 'target_state' input port specifies the state of the module we want to track. By default, there are three states: 'INIT', 'IDLE', and 'PROCESS', but users can define custom states as needed. The 'cycle_count' output port is used to indicate the number of cycles that the module has spent in the 'target_state', while the 'current_state' output port indicates the current state of the module. The 'current_state' is particularly useful for debugging hardware bugs, such as deadlocks, where the accelerator is stalled.

B. Software Integration

The hardware monitors are integrated with the target application through a new hardware profiling API in the SECDA-

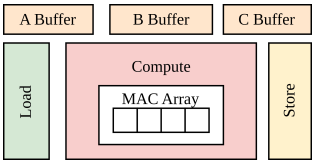


Fig. 2: Simple Matrix Multiplication Accelerator.

TABLE I: HM state profiles for the hardware unrolling. Each cell shows IDLE / PROCESS cycles. INIT state is omitted as it is zero for all runs. The final column reports the % difference of the PROCESS state from $2\times$ unrolling to $8\times$.

HM Module	$2\times$	$4\times$	$8\times$	Δ PROCESS
Load	633,861 / 33,817	524,581 / 33,871	410,685 / 33,781	-0.01%
Compute	402,406 / 265,520	459,353 / 99,248	397,426 / 47,408	-560%
Store	651,028 / 16,944	541,701 / 16,944	427,934 / 16,944	0.0%

Core library. This API allows the designer to configure the hardware monitors during runtime, such as changing the 'target_state' of a specific monitor to track different states of the module, and to read the 'cycle_count' and 'current_state' outputs of the monitors to analyze the performance of the accelerator and debug any hardware issues. We provide a simple driver implementation that abstracts away the low-level details of accessing the hardware monitors, enabling designers to easily perform hardware profiling.

III. Evaluation

For our experimental setup, we use the AMD KV260 FPGA board [3] (avaiable resource listed in Table II). We implement a simple matrix multiplication accelerator on the FPGA using the SECDA methodology [1], and integrate the hardware monitors into the design to track the performance metrics.

The accelerator consists of three main modules: the Load module, the Compute module, and the Store Transfer module. Figure 2 illustrates the architecture of this accelerator, where the Load module is responsible for loading the input matrices A and B from memory, the Compute module performs the matrix multiplication, and the Store Transfer module writes the output matrix C back to memory. The hardware monitors are integrated into the design to track the performance metrics, specifically, the cycle counts of the different modules of the accelerator. Additionally, we highlight the FPGA resource utilization with and without the hardware monitors to demonstrate the overhead of the monitoring infrastructure.

A. Performance Tracking

We compare the performance of different hardware unrolling factors within the MAC (Multiply-Accumulate) array, which is present in the Compute module, to demonstrate the capabilities of the hardware profiler in tracking performance changes due to design optimizations. Note that the unrolling factor determines the degree of parallelism in the MAC array, with higher unrolling factors allowing for more parallel computations, but also increasing resource utilization. For our evaluation, we use a fixed matrix multiplication problem size of M = N = K = 64.

Table I summarizes the three runs with unrolling factors of $2\times$, $4\times$, and $8\times$ for the MAC array. Increasing the unrolling factor improves the overall execution latency: the FPGA total decreases from $10,856$ ms at $2\times$ to $7,767$ ms at $8\times$, as reported by existing SECDA profiling.

The HM state profiles provide a finer-grained view of this behavior in Table I. The first state, INIT, is zero for all runs, so the meaningful comparison is between IDLE and PROCESS. Note that the Compute module exhibits the largest change in PROCESS-cycles, decreasing by $5.6\times$ from $2\times$ to $8\times$. In contrast, the Load and Store modules show a negligible PROCESS-cycle increase; this is because the unrolling optimization implemented in the Compute module. Overall, the HM profiles indicate that unrolling primarily shifts the Compute module state distribution, while Load and Store behavior remains comparatively stable.

B. Resource Utilization

Table II shows that the FPGA resource requirement with and without integrating HMs into the accelerator design. Taking the resource difference between the two configurations, we see that the monitoring infrastructure incurs minimal overhead, adding only 0.17% of total available FFs and 0.29% of total available LUTs, while BRAM and DSP resources remain unchanged. This demonstrates that the hardware profiler can be integrated with negligible impact on resource utilization, even when scaled to multiple modules with individual HMs.

TABLE II: Resource utilization comparison with and without hardware monitors for the matrix multiplication accelerator.

Configuration	BRAM 18K	DSP48E	FF	LUT
Without HMs	24 (8.3%)	201 (16.1%)	3680 (1.6%)	6657 (5.7%)
With HMs	24 (8.3%)	201 (16.1%)	4067 (1.7%)	6997 (6.0%)
Total available	288	1248	234240	117120

IV. Conclusion

We presented the SECDA Hardware Profiler, a lightweight runtime profiling approach for SECDA-generated accelerators. By integrating hardware monitors and a profiling API, the approach exposes module-level state behavior on FPGAs while introducing negligible overhead, adding only 0.17% of total FFs and 0.29% of total LUTs. Our case study shows that the profiler captures clear performance changes under different unrolling factors, making it useful for rapid hardware design analysis and debugging.

Acknowledgments

This work was partially supported by the UK Engineering and Physical Sciences Research Council (grant EP/R513222/1), the EU Project dAIEDGE (GA Nr 101120726) and the Innovate UK Horizon Europe Guarantee (GA Nr 10090788).

References

[1] J. Haris *et al.*, "SECDA: Efficient Hardware/Software Co-Design of FPGA-based DNN Accelerators for Edge Inference," in *SBAC-PAD*, 2021.

[2] IEEE, "IEEE Standard for Standard SystemC Language Reference Manual," *IEEE Std 1666-2011*, 2012.

[3] AMD. Kria KV260. AMD. [Online]. Available: https://www.amd.com/en/products/system-on-modules/kria/k26/kv260-vision-starter-kit.html