Insert & Save: Energy Optimization in IP Core Integration for FPGA-based Real-time Systems

Martin Geier
Chair of Real-Time Computer Systems
Technical University of Munich
geier@rcs.ei.tum.de

Marian Brändle
Chair of Real-Time Computer Systems
Technical University of Munich
braendle@rcs.ei.tum.de

Samarjit Chakraborty
Department of Computer Science
University of North Carolina at Chapel Hill
samarjit@cs.unc.edu

***Abstract*—Today, many industrial, automotive and autonomous systems like robots are deployed in high-temperature and battery-powered environments. Due to cooling and runtime, this limits the energy consumption and makes the design of such embedded real-time systems even more challenging. Though Field Programmable Gate Arrays (FPGAs) offer the required performance, their static and – load-dependent – dynamic energy consumptions continue to prevent a widespread adoption. The existing methods for dynamic power reduction (like clock gating) are either limited in savings or require disruptive changes to well-established FPGA design flows. Whilst the former is caused by optimizing on fabric level only, the latter is due to the lack of support for a more efficient (but not yet mature and standardized) high-level design entry in current tools. In this paper, we thus explore an optimization methodology based on an existing, but not-fully-utilized intermediate level of abstraction that emerges in the IP core integration phase of the design. To this end, we exploit the fact that the vast majority of FPGA-based real-time processing pipelines is not exclusively assembled using a single type of design entry – i.e., neither entirely hand-written nor high-level synthesis only. Instead, suitable IP cores (from a variety of sources) are integrated via standardized bus interfaces such as AXI, Avalon or Wishbone. To facilitate effort- and power-efficient clock gating on integration-level, we present two "insert and save" IP cores that harness application information extracted from current AXI3 and AXI4-Stream interfaces. Based thereon, both cores precisely control the clock signals of every downstream processing stage for maximum energy savings. This approach not only nicely integrates with today's predominantly AXI-based designs but also results in clock gating structures that are particularly suitable for current FPGAs – as demonstrated by experimental evaluations on a Zynq-based Visual Servoing System with energy savings of 26%.**

I. Introduction

The sustained need for higher communication and computation densities continues to yield increasingly complex system architectures. Besides rising the costs for design and manufacturing, those developments also affect the overall energy consumption, causing additional challenges to power and cool the device. For Real-time Systems (RTSs) that are bound to predefined latency constraints, energy efficiency traditionally has not been a major concern. Recently, however, the advent of autonomous systems with their often battery-, space- and heat-constrained operating environments has intensified the need to carefully balance cost, functional-temporal aspects and energy consumption during all design phases. Even though power-only management strategies are well researched for both stationary and mobile systems [1], [2], their application in safety-critical domains – such as avionics, industrial or automotive – requires additional care to avoid, or at least limit, adversely affecting the temporal RTS behavior. Typical deadlines range from hundreds of µs to several ms, and issue from the dynamics of the physical process [3]. Therefore, most closed-loop control systems cannot tolerate the additional delay caused by, e.g., Dynamic Voltage and Frequency Scaling (DVFS) [4], data buffering [5] and other system-level methods. This is because additional or, often worse, variable delay – i.e., jitter – invalidates previous assumptions about crucial temporal RTS characteristics like its *input-to-output latencies*. The latter capture the overall delay from arrival of a sensor value until the corresponding actuation signal is computed/sent by the system. Their inherent negative impact on the closed-loop performance is usually compensated during control design [6], and assumed to be constant. In contrast to relatively new sensor-event-driven approaches, traditional control strategies are often based on the time-triggered *periodic activation* of each control task mapped to the RTS. To simplify the formal design process and increase the stability of the closed control loop, the deadline of the task commonly is fixed to a value smaller than its period [7], [8]. As this and further RTS choices often result in *slack* (i.e., stages of the processing pipeline being temporarily idle, cf. Sec. II-B), it also creates an *opportunity for energy optimization* – e.g., after deadlines. To avoid physical instability such as oscillation, it is, however, crucial not to interfere with the RTS's temporal – and thus indirectly functional – aspects during energy optimization.

Existing methods thus either aim at (often system-level) co-optimization/verification, or reduce the energy consumption on a much lower and thus less effective level of abstraction. Apart from traditional, software-driven Systems-on-Chip (SoCs), this also holds true for heterogeneous RTSs that combine a flexible, but energy-inefficient Field Programmable Gate Array (FPGA) fabric with a fixed-function SoC. With the latter containing one or even more highly efficient Central Processing Units (CPUs), memory controllers and I/O peripherals, such a Programmable SoC (pSoC) is particularly suitable for high-speed mixed-hardware/software real-time pipelines, often found in industrial, automotive and autonomous application contexts [9]–[11]. Albeit such FPGA-based architectures provide the required communication and computation capabilities, the flexibility of their run-time-programmable fabric comes at a cost. Firstly, their design flows are considerably more complex than traditional software-only development – due to the additional effort in specification, implementation and integration of custom, application-specific processing blocks in hardware. Such Intellectual Property (IP) cores come from various sources such as FPGA/pSoC vendors,

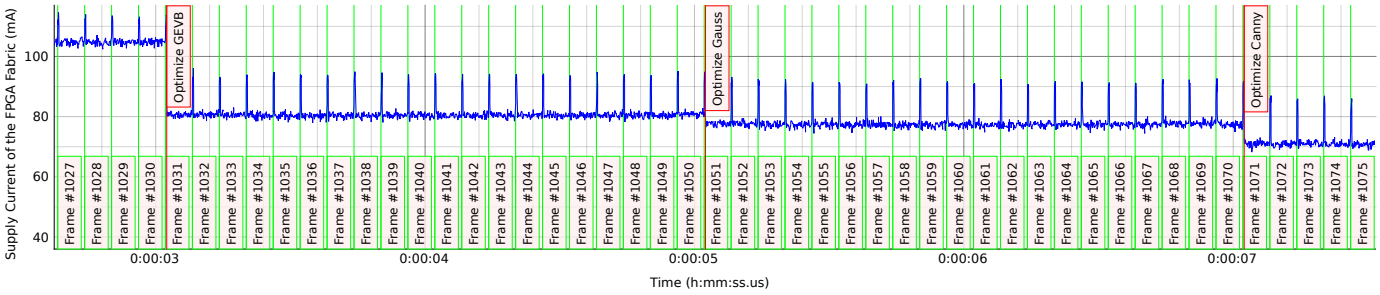

Fig. 1. Achieved Savings in our Vision System with IP Cores for Image Acquisition (GEVB), Smoothing (Gauss) and Edge Detection (Canny): 23%, 3% and 6%

3rd-party companies, high-level synthesis (HLS) tools or hand-written Hardware Description Language (HDL) code [12], and predominantly incorporate standardized bus interfaces (such as ARM's AXI [13], Intel's Avalon or Wishbone). Based thereon, the system designer selects suitable IP cores to create the hardware subsystem of the real-time application pipeline. Their bus interfaces not only simplify the core-to-core integration (within the FPGA fabric), but also connect the hardware pipeline to the software-based remainder of the RTS, i.e., its CPUs, memories and I/O peripherals. Although standardized bus interfaces such as AXI3 for memory-mapped and AXI4-Stream for, e.g., video ports thus serve as a helpful "lowest common denominator" (as detailed in Sec. II-C), the subsequent integration/verification of such heterogeneous embedded or RTSs remains a difficult task.

As a second disadvantage, today's FPGA fabric still requires considerably more energy than a functionally equivalent, fixed-feature Application-specific Integrated Circuit (ASIC) [14] due to the inherent implementation overheads for logic and routing. Compensating for this, FPGA vendors continuously implement new power management features in the device fabrics (Sec. II). Currently, however, these hardware improvements are not fully utilized in the majority of FPGA-based RTSs, as the associated offline design and (online) management techniques are limited. As detailed in Sec. II-A, runtime solutions such as D(V)FS and (partial) power gating cause latencies beyond what many RTSs are able to tolerate. Design-time approaches, like automatically inferred or HLS-based register enables, on the other hand, only yield relatively small savings and require disruptive changes to the established, decentralized FPGA design flows, respectively.

Proposed Methodology: In this paper, we thus investigate a level of IP abstraction that enables an intermediate clock gating methodology for modern FPGA-based RTSs. It utilizes already existing medium-grained power management capabilities without major changes to established design flows, and exploits the precise application information intrinsically contained in most, e.g., AXI-based, standardized bus protocols used in today's IP-driven hardware pipelines. We present two "insert and save" IP cores that can easily be added to current designs with memory-mapped AXI3 and/or less complex, e.g., video-carrying AXI4-Stream (AXI4S) interfaces. By tracking individual transactions on the respective bus ports and carefully manipulating the flow control and data signals, our IP cores are able to precisely limit the switching activities of all the connected downstream clocks and hardware pipeline IPs. Such *integration-level* clock gating has many advantages over traditional (high- or low-level) techniques that render it particularly applicable to today's complex, mixed-hardware/software processing devices. Firstly, thanks to their efficient design, our IP cores neither require a lot of fabric resources, nor have significant effect on the pipeline's temporal behavior – thus avoiding a functional modification of a system. Secondly, this approach directly integrates with the established IP-based designs flows (as our cores are simply added to the IP catalog of the respective tool) and is suitable for (FPGA-based) embedded *and* RTSs as well. Lastly, the chosen level of "clock gating abstraction" maps rather nicely to the (medium-grained) FPGA clocking resources. All this can be seen in our extensive experimental evaluations on a Xilinx Zynq pSoC, for which we apply this methodology to optimize the energy consumption of a real-world mixed-hardware/software Visual Servoing System (VSS). Driven by a high-speed GigE Vision camera, hardware-accelerated image acquisition/processing stages and a software controller enable the RTS to stabilize a magnetic levitating ball used as a reference application scenario for complex VSSs. By applying the clock gating methodology to the VSS's individual IP cores (for image acquisition and processing), we are able to significantly reduce the dynamic power, as evident from Fig. 1. As the absolute savings scale with a pipeline's complexity, this approach is highly beneficial to constrained autonomous RTSs, while only barely affecting fabric performance and utilization. In summary, the **main contributions** of this paper are

- *Integration-level Clock Gating* for immediate reduction of dynamic energy consumption in current FPGAs & pSoCs,
- Design and Implementation of *Clock Gating IP Cores* for memory-mapped AXI3 and stream-based AXI4S IPs, and
- Evaluation of achievable *Power Reduction* in a real-world VSS with the derivation of a *Savings Model* for our cores.

Outline: The rest of this paper is organized as follows. Technical background and related work are presented in Sec. II – also including a comparison to our solution and a discussion of vital aspects in control design and typical hardware architectures for today's embedded or real-time systems. Based thereon, Sec. III infers the underlying principles and discusses key properties of integration-level clock gating on different bus interfaces. Then, Sec. IV-A describes crucial features of modern pSoCs (using a Xilinx Zynq as an example), while Sec. IV-B touches upon the VSS scenario. Implementation details and evaluation results of our two IP cores for energy optimization of the real-world VSS are given in Sec. V, including our IP utilization- and size-based power savings model. We finally conclude our work in Sec. VI.

II. Background and Related Work

Besides the considerably larger development effort (in contrast to a software-only platform), FPGA-based systems suffer from inherent implementation overheads of logic circuits (cf. Sec. I), yielding higher energy consumption than a fixed-function RTS. Their flexibility, short time-to-market and, compared to ASICs, insignificantly low setup costs, however, continue to accelerate FPGAs into formerly unreachable application domains. Simultaneously, power management features steadily improve on the fabric-level (to be combined with suitable design and/or online techniques for maximum efficiency). Current FPGA fabric thus incorporates automated power gating of unused memories [15] to control their static leakage. The dynamic power, by contrast, is reduced via scaling advances in chip fabrication, support for medium-grained clock gating [16], and automated inference of register(-set) enables [17]. Although numerous (predominantly high-level) techniques have been proposed over the years [18]–[25], these already available power management capabilities of FPGAs are currently deployed neither systematically nor to the extent already technically possible. Apart from their (potential) impact on the RTS's temporal behavior, this is because most of the high-level techniques – intrinsically – require that the entire design (or at least a large part thereof) is under the control of a single central, but not (yet) established tool. Besides the higher risk of (unidentified) tool issues, this centralization breaks with the traditional IP-based design flow, thus complicating not only the initial integration but also testing/verification. For low-level solutions (e.g., inference of register enables) already integrated into commercial design tools, on the other hand, this also holds true as their lack of application awareness limits the achievable energy savings, albeit without significant impact on the timing.

This work, in contrast, specifically uses the medium-grained (i.e., global and/or regional) clock gating capabilities of current device fabrics to comprehensively reduce the energy consumption of FPGA-based RTSs (Sec. II-A) with fixed and negligible latency and resource overheads. Both proposed IP cores utilize the idle time often found in a mixed-hardware/software control loop (Sec. II-B), and nicely integrate with the standardized bus interfaces in SoC-, FPGA-, or pSoC-based systems (Sec. II-C).

A. Energy Consumption & Management Techniques in FPGAs

As dynamic energy consumption of CMOS circuits depends on supply voltage and switching activity, ASIC designers heavily rely on a variety of established methods to reduce either one or both factors – with a lower voltage also decreasing the static leakage. While only a few techniques such as DVFS thus limit both static and dynamic power, others are focused on lowering the switching activities on clock and/or data signals alone, with intrinsically no effects on leakage currents. Nevertheless, clock gating, register enables and various other online techniques for reduction of dynamic power are widely used in current, heavily optimized fixed-function devices. There, necessary additions to the circuitry are included during the ASIC design process, and thus cause only minimum area overhead. For FPGAs, however, only a subset of methods is available, as their fabrics cannot be customized to the same (i.e., transistor-level) degree as ASICs. Instead, designers and tools are restricted to the capabilities the respective FPGA architecture provides. For modern fabric, this includes Dynamic Partial Reconfiguration (DPR) and, recently, multi-level clock gating (CG) ranging from global and regional clock buffers to per-resource (e.g., few registers) clock enables. Architecturally, there is a key difference between those two CG extremes – based on whether they influence an entire clock tree (spanning across significant area of the device) or an individual resource like the eight single-bit registers in a Xilinx slice [26]. Although less customizable than on an ASIC, medium-grained CG has been reported to enable similar dynamic power savings in FPGAs [27]. Details of the clock distribution in current, e.g., Xilinx Zynq pSoCs and other internals are shown in Sec. IV-A.

In addition to low-level optimizations already integrated into current synthesis tools (such as automated inference of register enables), the DPR- or CG-based high-level techniques fall into the following main categories. DPR depends on predefined and pre-synthesized IP cores (e.g., hardware accelerators) that then are repetitively swapped in and out of the FPGA. This can save energy (either by disabling the fabric entirely or by using some parts of the device for multiple IPs sequentially instead of implementing all stages in parallel) and intrinsically requires application-level design and runtime control. As reconfiguration of even relatively small accelerators requires several ms, however, *DPR-based power management* significantly impacts temporal system behavior [18]. If then complemented with power gating (for even higher savings), the fabric's turn-off/on times have to be factored in as well [19], which further reduces applicability, as many RTSs cannot tolerate the resulting latencies. In case of high-rate control systems, application deadlines are in the same order of magnitude (due to similarly short periods, e.g., 5.6 ms for typical VSSs as shown in Sec. IV-B), thus prohibiting DPR.

Software-controlled CG, as another high-level approach, has similar limitations w.r.t. application timing constraints, as each CPU-based (de)activation using memory-mapped I/O (MMIO) writes has a comparable delay [20], [21]. It, in addition, is only applicable if the software runtime environment has some "prior knowledge" of when CG may be applied. While this holds true for traditional hardware accelerators that are only used by CPU software, any I/O interfaces implemented in the FPGA (such as I/O peripherals like Ethernet controllers) have to be operational at all times as incoming data are lost otherwise. As the runtime environment is unaware of any asynchronously arriving data, it is unable to reactivate the I/O controller's clock when required.

Automatically inferred CG, however, is mostly controlled by enable signals generated within the FPGA hardware itself. This holds true for automatically inferred clock enable (CE) signals, as locally produced by current implementation flows, and those derived by selected HLS tools. In the former – low-level – case of, e.g., Xilinx' "Intelligent Clock Gating" solution [17], small logic-only circuits to compute the CE signals are automatically added during the place-and-route phase of the flow. Automated analysis of logic equations is used to identify both the registers whose outputs are not always processed further and the control signals determining that. Based thereon, the CE inputs of every suitable slice are used to reduce downstream switching activity.

More "application-aware" CE logic, on the other hand, may be explicitly created by dedicated HLS design tools often used on top of the traditional synthesis, place and route steps [22], [23]. While such fabric-controlled CG yields negligible (i.e., single-cycle) delays and thus not impacts RTS latencies, this approach only reduces the energy consumption of the cores created *within* such a tool. As the conversion of existing IP libraries to HLS would not only require considerable efforts but also disrupt the established design, simulation, implementation and verification flows, HLS-based CG, currently, only has limited applicability.

Asynchronously controlled CG for ASICs possibly combines low latencies and traditional IP-based design by inserting local, distributed CG controllers [24]. Driven by asynchronous handshake signals, latch-based CG blocks dynamically determine if the downstream logic can be disabled. Their use of latches and inverted clock signals, however, restricts this solution to ASICs as such circuits may not be implemented for current FPGAs. In addition, the proposed controller appears to support only single read/write accesses at a time – rendering it unusable for today's complex interfaces that support many outstanding transactions. The counter-based alternative proposed in [25] remedies this as evident from its schematic (cf. Fig. 8 on Sheet 8 of [25]), albeit again targeting fixed-function ASICs/SoCs. In addition, it aims at CG between the bus master(s) in the system and any, directly connected (synchronous or asynchronous) bus bridge functions such as rate/width converters, memory management unit, or an interconnect slave port. This may restrict the CG to a relatively small fraction of the system – thus limiting achievable savings.

It should be noted that, despite steadily increasing flexibility of current FPGA fabrics and design tools, a fixed-function RTS may benefit from additional energy optimizations such as core-internal, clock-based *automatic wake-up* (e.g., using local CG). Due to lack of fabric support for arbitrary clock routing, IPs for FPGAs could only implement this via the medium-grained CG resources (Sec. IV-A). As evident from their utilization reports, however, this is not the case, leaving only manual instantiation.

Relation to the Proposed Methodology: Contrastingly, our integration-level CG has negligible impact on the application's temporal behavior and conforms to established IP-based design flows. Once added (using, e.g., Xilinx' IP Integrator), our cores are capable of tracking bus activity on fully-featured AXI3 and AXI4S interfaces, and enabling a downstream clock with fixed, single-cycle delay. They thus neither impact timing nor restrict RTS designers w.r.t. flow, and, moreover, facilitate a significant reduction of the dynamic power for all AXI3- or AXI4S-driven IP cores with known or observable – or no – tail latencies. This extends our methodology beyond CPU-controlled accelerators, as demonstrated for the purely FPGA-driven image acquisition of the VSS (cf. Sec. V-A1). Compared to (even less obtrusive) inference of register-level CEs, our IP cores yield higher power reductions (due to their integration-level application awareness and their capability to gate entire clock trees), and have a lower logic overhead. Unlike above ASIC implementations, however, they insert fixed single-cycle delays, or buffer the data path. As with other CG-based solutions, and in contrast to DPR or power gating, reducing the static leakage also remains out of reach.

B. Closed-loop Control: Design & Implementation Challenges

Even before digital computer systems were readily available like today, closed-loop control was widely used to stabilize and guide physical processes (by either mechanical means or using analog circuitry). Based on one or multiple *sensors*, the current state of the process, also known as plant, is captured. A control algorithm then converts these sensor signals and predetermined set-values/trajectories into an output signal by implementing a fixed, or even adaptive *control law*. Designed to minimize both the difference between actual and desired states (i.e., the *error*) and, often, the energy spent to do so, the output signal prompts an *actuator* to implement the computed control action onto the physical process. The resulting feedback-driven changes to the plant state finally close the control loop and enable its designer to compensate instability or disturbances in the physical world.

Establishing the appropriate control law(s) for one particular plant, however, can be a challenging process even without real-world complications such as sensor noise or actuation limits. In addition to capturing the properties of the physical process into a plant model (i.e., initial *system identification*), the subsequent *control design* phase also may not be sufficiently tractable with the established linear methods for which closed-form solutions are available. If control problems at hand are beyond such, e.g., linear-quadratic (LQ) or proportional-integral-derivative (PID) controllers [28], [29], complex non-linear approaches are used, which often rely on numerical (i.e., non-closed) computations. In both cases, the addition of the feedback path from plant (via sensor) to controller and back to the plant (via actuator) has the potential to get any *otherwise stable* physical setup to oscillate.

Even in time- and value-continuous systems (using mechanical or analog feedback), the **stability** of a plant with controller often is difficult to ensure due to the unstable and/or non-linear nature of the physical processes alone. As additional challenge, implementing a perfectly stable (mechanical/analog) controller using a digital computer system can cause new instabilities that render the system unusable. Apart from *discretization* issues in terms of sampling events and signal values, such digital control systems intrinsically add a *delay* for sensor acquisition, control computation and output of an actuation signal. Even though the former can be remedied with, e.g., high-rate/resolution analog-to-digital/digital-to-analog converters (ADCs/DACs), the latter effectively reduces the remaining phase margin of the loop and thus its stability [8]. During the *offline control design* step, this input-to-output latency will therefore be compensated [6], e.g., by increasing the gains of the control law (which yields a more aggressive actuation behavior also reducing overall resilience). As this (formal) design of the desired control law and its actual implementation in form of the controller mapped to a computer system are regularly performed independently, constant latency values are thus selected and used. Any deviation thereof during the execution of the controller then causes a *mismatch* between the projected and actual closed-loop performance, which limits the applicability of effective energy reduction tools (Sec. II-A).

Like many of their embedded (i.e., not so tightly constrained w.r.t. delay) relatives, the majority of RTSs follows an inherent

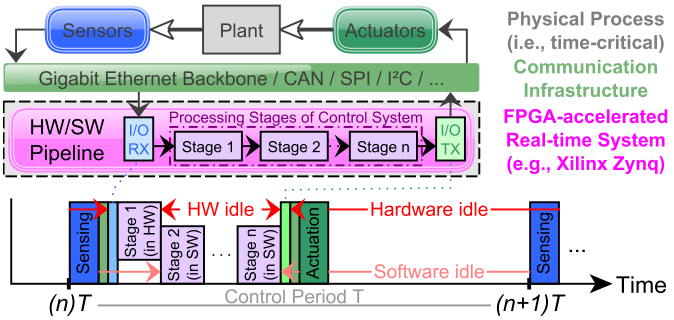

Fig. 2. Typical Control System: Physical World (Plant, Sensors and Actuators; top) and FPGA-based Real-time System (center) with an exemplary Execution Schedule of the Processing Pipeline and Idle Times within a Period T (bottom)

input-process-output structure to, e.g., close a control loop. For highest-possible predictability of its timing, resource usage and the resulting control performance, all critical tasks (i.e., sensor acquisition, computation of control law and output of actuation signal) are predominantly performed periodically. Such a *time-triggered* approach is well supported by a variety of scheduling algorithms in real-time operating systems, although period and priorities in the entire system still have to be selected carefully.

Fig. 2 shows a typical (networked) control system consisting of the physical world (with plant, sensors and actuators) and an FPGA-based RTS (top and center, respectively). Its functional-temporal behavior is as follows, and also seen in the exemplary execution schedule in Fig. 2 (bottom). The sensed plant state is received (RX) by the RTS over a communication infrastructure and, once read by an input task (I/O RX, light blue), fed to the actual processing stages. Except for measurement delays in the sensor (e.g., camera) itself, the total latency up to this point can often be neglected due to large plant inertia and long execution times of the subsequent hardware/software tasks. Each of these processing stages (light purple, Fig. 2 center/bottom) performs a vital function towards implementing the control law – such as sensor acquisition, filtering or control computation – and therefore gets individually mapped to either hardware or software of the RTS. Based on the transmitted (TX) actuation signal issued from an output task (light green), the loop is eventually closed. It can be clearly seen that this results in significant **idle periods** (i.e., slack), in particular for every hardware-accelerated stage, as it is only active for a short fraction of each period and as underlying fabric resources cannot be reused by DPR (Sec. II-A).

In case an *event-triggered* control/execution scheme is used, its inherent property to compute and actuate as rarely as (upper and present) error values allow commonly results in longer idle phases compared to its less adaptive, time-driven alternative. In summary, both time- and event-triggered systems typically idle for a non-zero amount of time and within at least some of their hardware processing stages/units. The resulting opportunity for energy optimization, however, has to be seized on the per-stage basis for maximum savings, and without adding any significant processing delays that would degrade the control performance. For FPGA- and – if factored in before the costly manufacturing step – even ASIC-based systems, our IP cores implement just that, although their actual deployability depends on the number of bus interfaces and the underlying general hardware pipeline.

C. Typical Architecture & Bus Interfaces of Real-time Systems

The hardware architectures of modern embedded or real-time systems typically comprise various communication, processing and storage elements as shown in Fig. 3. For generic (e.g., non-video) data, standard I/O controllers (such as CAN, Ethernet or USB cores) and computational resources like traditional CPUs, Graphics Processing Units (GPUs) with General-purpose (GP-GPU) computing capabilities or, recently, artificial intelligence accelerators are integrated via bus interconnects and memories. Predominately, a large, central (e.g., DDR) memory containing countless independent data buffers serves as an exchange point for all system components. In traditional software-driven SoCs without an FPGA fabric, the individual buffers are managed by an Operating System (OS) running on the CPUs – and assigned to other components via a Hardware Abstraction Layer (HAL). Based thereon, I/O peripherals (such as USB and Ethernet IPs), accelerators (e.g., GP-GPUs) and software tasks running on the CPUs exchange the application information using memory and MMIO transactions to and from the data buffers independently.

To facilitate the use of such a central memory and reduce the design and verification effort, the vast majority of architectures relies on standardized bus interfaces between components, e.g., ARM's AXI, AXI-Lite or AHB definitions. Such interfaces for *memory-mapped transfers* are necessary on all aforementioned system components (cf. cyan links in Fig. 3) and carry address, data, and control information. A master device (such as a CPU) initiates transactions by defining transfer direction (read/write), desired address (range), and additional parameters such as beat width and alignment. Slaves (e.g., memories) then respond via data and/or control signals, whereas interconnects link multiple masters and slaves by arbitrating the incoming transactions. As a crucial difference between write and read transfers, the latter require at least two phases transmitting, firstly, an address from master to slave and, secondly, the slave's response carrying the requested data. For writes, address and data are commonly sent simultaneously – leaving only the slave's acknowledgement for later transmission. With bus interfaces that support outstanding transactions (i.e., issuing another request before the current one has finished), however, the transfers of associated address, data and acknowledgements become interleaved. This not only adds considerable complexities to masters, interconnects and slaves, but also makes monitoring the interface states/activity difficult. Nevertheless, the integration and performance advantages have made a small number of complex but standardized bus systems the predominant solution for memory-mapped interfaces today.

A similar situation exists for specialized stream-based – e.g., video – acquisition, processing and output pipelines commonly found in high-performance RTSs (Fig. 3, top). Due to the deeply parallel nature of input data and processing algorithms from, e.g., camera, radar or lidar sensors and applications, traditional memory-mapped I/O and computational resources are often no longer sufficient to meet the performance goals. Thus, purpose-built application-specific pipelines are used that heavily rely on *stream-based transfers* between their individual components as indicated by the corresponding interfaces in Fig. 3 (pink links).

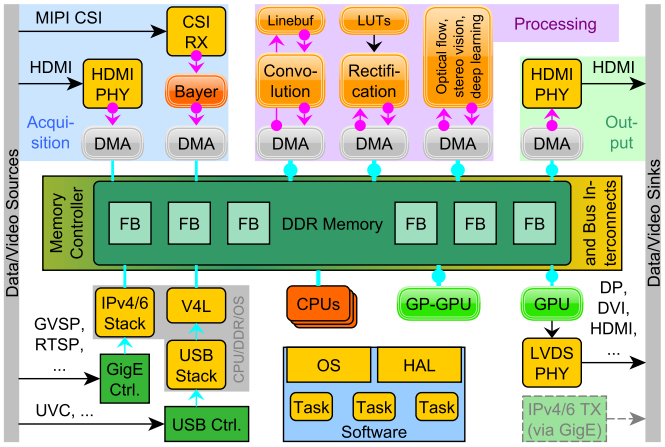

Fig. 3. Typical System Architecture: Acquisition, Processing and Output Units integrated via memory-mapped (cyan) and stream-based (pink) Bus Interfaces, with all possible Locations for our CG Controllers indicated by cyan/pink Dots

In many cases, they combine multiple back-to-back processing stages to form an in-stream pipeline that (e.g., in case of video) operates on single pixels or image rows (stored in a linebuffer). This, for instance, often holds true for image acquisition blocks (such as Bayer demosaicing, gamma correction and scaling) or convolution-based (2-D) algorithms like Gaussian filtering [9]. More complex image processing steps such as rectification (for subsequent stereo vision), however, depend on the entire image to be available in a Frame Buffer (FB) for non-sequential reads predefined by Look-up Tables (LUTs). Each conversion from a stream-based to a memory-mapped interface or vice versa then requires Direct Memory Access (DMA) controllers connecting the pipeline to its associated FBs (Fig. 3, gray and light green).

III. Proposed Methodology for Energy Reduction of FPGA-RTSs via Integration-Level Clock Gating

In the foreseeable future, the design, test and integration flows for FPGA-based embedded and RTSs will remain driven by all kinds of IP cores that are linked via standardized bus interfaces as introduced in Sec. II-C. Whether an entirely HLS-generated, unified and globally optimized system design might be feasible is not yet clear. Based on this and the additional insight that the dynamic energy consumption of the underlying fabric remains significant (Sec. V-B), we argue that an **intermediate solution**, combining traditional IP-based design with *higher-level energy optimization*, is required. Therefore, we propose to consider all the deployed, standardized bus interfaces as potential locations to insert a small, *decentralized CG controller* using established and proven IP-based design flows. Once looped into the chosen link, a protocol-specific bus snooping logic decodes all passing transactions to determine whether any write or read requests to its downstream slaves are still pending. The system designer is then provided with a precisely controlled, gated clock signal to be fed into the downstream core(s) of choice. By disabling this clock whenever possible, a *significant reduction in the dynamic energy consumption* can be achieved. As the CG controllers are instantiated at the same time and phase of the design process as all other IP cores, such *integration-level clock gating* combines application-aware power management and the established flow.

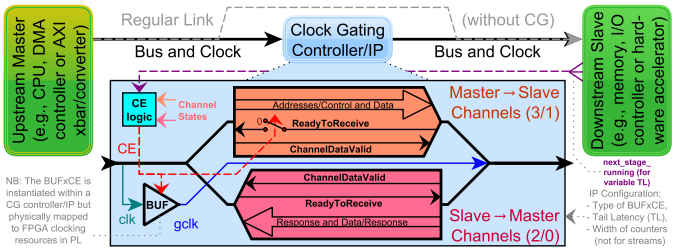

Fig. 4. Generic Bus Interface between Master (left) and Slave (right), Internals of Bus Channels and CG Controller (center) and Synchronization Mechanisms

The methodology is applicable to both memory-mapped and stream-based interfaces with downstream IPs not continuously used, and *covers most links* in typical systems, as shown by the candidates denoted by cyan or pink dots in Fig. 3. Even though there are numerous possibilities, the CG controllers (currently) are inserted manually – although an automated identification of meaningful "CG islands" based on bus interfaces, clock signals and fabric usage of IPs is feasible. Their in-depth knowledge of the application pipeline at hand, however, enables experienced system designers to arrive rapidly at an optimized CG solution. Due to the extremely low resource and latency overhead of our cores (Sec. V-C), their bypass feature (Sec. V-A), and the large number of CG-capable local clocks available in current FPGAs (cf. Sec. IV-A), moreover, it is entirely feasible to assign one of our CG controllers to *every* applicable link. Such a (temporary) FPGA design then may easily drive a systematic exploration of the achievable savings for given architectures in one single run.

Fig. 4 illustrates integration-level clock gating by means of a generic bus interface composed of, at least, one master-to-slave communication channel (orange), which holds true for the vast majority of today's systems. Besides its (address, control/status or actual data) *payload*, each channel also intrinsically includes a set of *handshake signals*. While *ChannelDataValid* indicates to the receiving side that the payload signals are currently valid on this channel, *ReadyToReceive* informs the transmitting node whether the receiver is accepting or consuming new payload at this time. Together, they coordinate the synchronous transfer of payload across channels – either from upstream to downstream device (master-to-slave, orange) or vice versa (slave-to-master, light red). Each *ReadyToReceive* also serves as a means of flow control and enables the receiving side to slow the transmissions via a backpressure mechanism. While, for instance, (memory-mapped) AXI3 comprises three master-to-slave plus two slave-to-master channels, a (stream-based) AXI4S interface only has one master-to-slave channel, as detailed in Sec. V-A1/3 or [13].

Normally, a master is connected directly to its slave(s), often accompanied by a shared, common clock (Fig. 4, dashed gray). Once placed in between, our resource-efficient **CG controllers** exploit these *already existing* handshake signals in two ways to generate a dedicated, gated clock for their downstream core(s). On the one hand, the states of both the master-to-slave (orange, cf. Fig. 4) and slave-to-master (light red) channels are analyzed by a small CE logic within our cores (cyan), which detects any bus activity based thereon and counts outstanding transactions. On the other hand, all master-to-slave channels (of which there are three/one in AXI3/AXI4S, respectively) are rather carefully

altered by adding a multiplexer to each *ReadyToReceive* signal. As shown in Fig. 4, these simple switches are controlled by the CE logic (red wire) and used to de-assert each *ReadyToReceive* from slave to master if CG is currently enabled. This way, each newly arriving transaction is stalled until the CE logic activates the gated downstream clock (gclk) by re-enabling the included buffer (BUF) in the FPGA's clock tree (cf. Sec. IV-A/V-A). As the CG cores require only one single clock cycle for activation, this results in a **known, constant and negligible delay** of, e.g., 8 ns for a typical rate of 125 MHz (as in the VSS, cf. Sec. V-C).

The reverse transition from an active to the clock-gated state only occurs after all the bus transactions are completed and the downstream logic is no longer active. The latter is reflected via a fixed or variable *tail* (i.e., post-transaction) *latency* – which is either statically determined (and stored in the CG controller) or dynamically captured by means of a "running" signal from any downstream IP. With our CG cores implementing this extended handshake/packpressure mechanism, a master is stalled instead of accessing unreachable slaves – which, conversely, inhibit an early shutdown of their gated clock as long as remaining active (despite all interface channels already reaching their respective standby states again). In other words, CG is applied *if and only if* no bus transactions are pending *plus* the downstream logic is idle – enabling a *fine-grained, per-cycle activation* of the entire (now dedicated) clock tree with all associated fabric resources. The methodology at hand thus combines significant reductions in (dynamic) energy consumption, low resource usage and ease of use with deterministically low impact on the timing, making our CG cores an **efficient and safe** solution for IP-based flows.

It thus is *ideal for all latency- and energy-constrained RTSs*, which not only require latency-*neutral* energy optimization but also "by design" exhibit significant *idle phases* to be exploited. This slack, on the one hand, is a result of the control designer's choice to increase stability by selecting a deadline shorter than the period. On the other hand, the complex sensing and control application pipelines required today heavily depend on FPGAs to implement a suitable mixed-hardware/software architecture, which also causes slack due to the sequential execution of each task comprising the control pipeline (cf. Fig. 2). Particularly in case of processing stages mapped to hardware (i.e., the FPGA's fabric), the accumulated idle slack of the various bus interfaces (Sec. II-C) and associated IP cores implies substantial potential for energy savings. The plant's dynamics and sensors therefore (indirectly) define the best case, which thus is hard to estimate.

Implementation details and evaluation results of our generic AXI3/AXI4S CG cores are given in Sec. V. It should be noted that they are not only applicable to RTSs but any sort of system mapped to current FPGAs or, if adapted accordingly, to ASICs.

IV. Reference Platform and Application Scenario

We use the Xilinx Zynq as one of the two major pSoC architectures currently available to introduce their SoC/fabric internals. The Zynq enables a high-speed, mixed-hardware/software VSS serving as our reference application, hosted by a ZC702 board. Besides pSoC (Sec. IV-A) and real-world use case (Sec. IV-B), we also briefly introduce our measurement system (Sec. IV-C).

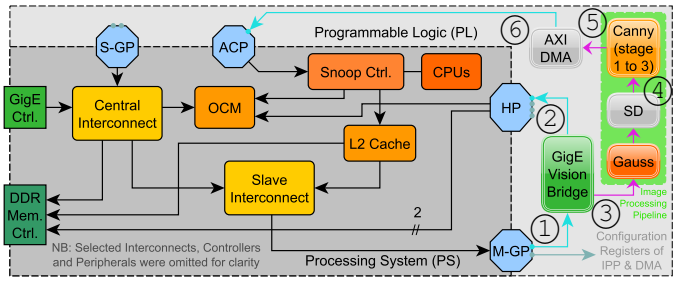

Fig. 5. Visual Servoing Scenario on Zynq: PS I/Os (left), PS (center) and PL with hardware-based Image Acquisition/Processing & AXI Interfaces (①–⑥) implementing memory-mapped (cyan) and stream-based (pink) Data Transfers

A. Zynq: Processing System, Programmable Logic & Internals

Like their Intel counterparts, Xilinx Zynq pSoCs combine a fixed-function, CPU-driven SoC with flexible FPGA fabric. On the medium-sized Zynq 7Z020 used in our evaluations, a dual-core ARM Cortex-A9 with shared L2 cache, On-Chip Memory (OCM), various I/O peripherals and bus interconnects form the Processing System (PS) as shown in Fig. 5 (dark gray). It relies on four general-purpose (GP), four high-performance (HP) and one ACP (Accelerator Coherency Port) *AXI3 interfaces* to link to the FPGA-equivalent Programmable Logic (PL). While two Master GP (M-GP) ports enable transfers from PS components to IP cores in the PL (light gray), the Slave GP (S-GP), HP and ACP interfaces serve the opposite direction (i.e., PL to PS). A variety of fixed-function I/O cores with interrupt request (IRQ) and (in selected cases only) DMA capability are integrated into the PS, and (due to their low energy consumption) often help to connect the RTS to its outside world. This includes two Gigabit Ethernet (*GigE*) *controllers* that use Scatter-Gather (SG) DMA to transfer incoming and outgoing data to and from the off-chip DDR memory, respectively. In the receive direction (crucial for the VSS's hardware-based image acquisition in Sec. IV-B), the detailed mode of (SG/DMA) operation is as follows. At initialization, the CPUs allocate a linear region in DDR memory that contains addresses and utilization status of 256 receive buffers. The GigE controller continuously searches this list for the next unused receive buffer and saves its respective memory address. After reception, the frame payload is then stored at the location identified before – which is tagged as "used" for later retrieval. Incoming Ethernet frames are thus *scattered* to non-sequential, potentially changing receive buffers without CPU intervention, which improves performance and reduces energy consumption. Details of GigE controllers and SG/DMA are available in [30].

Apart from Configurable Logic Blocks (CLBs), Block-RAM (BRAM) and DSP slices, current *FPGA fabric* features various dedicated resources for clock generation and distribution [16]. Depending on the size of the PL, Zynq devices contain up to 8 clock management tiles capable of synthesizing frequency- and phase-shifted signals (from dedicated clock inputs). In addition to those, up to four "fabric clocks" (`FCLKx`) are available from the PS. Together, they feed 32 *global clock lines* – reaching the entire device – albeit only 12 can be used within a single clock region (CR). As each CR spans 50 CLB rows and supports four additional *regional clocks*, the number of clocks increases with the device (i.e., PL) size. Depending on the used clock sources,

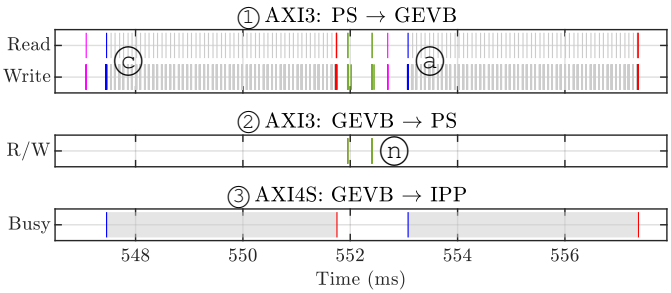

Fig. 6. Bus Transactions on AXI Interfaces of GEVB over 2 Control Iterations

thus at least 16 independent clock signals are available per CR. Due to restrictions of the clock management tiles (CMTs) that, amongst others, drive the regional clocks, only 12 clock signals can be gated by means of a horizontal clock buffer (BUFHCE) in each CR. Compared to earlier architectures, however, such a medium-grained CG capability is a significant improvement as designers (or tools) no longer have to use precious global clock lines to reduce the switching activity of one, localized IP core. For complex logic structures that span across multiple CRs and thus cannot be gated using one single BUFHCE, a global clock signal has the advantage that CG can be implemented using the associated upstream buffer (BUFGCE) for even higher savings. On the local (e.g., CLB/BRAM) level, individual, *per-resource CEs* are available for mostly manual CG – as tooling is limited.

B. VSS: Image Acquisition, Image Processing & Control Loop

To evaluate the achievable savings of integration-level CG in a realistic real-time application scenario, we draw upon a high-speed, mixed-hardware/software VSS implemented on a Zynq-based RTS. The control goal is to maintain the stable levitation of a metallic hemisphere, located below a configurable electromagnet. An industrial *GigE Vision camera* continuously, i.e., at 178 frames per second, records the sphere's position – in terms of a grayscale image transferred via the GigE Vision Streaming Protocol (GVSP) [31]. Once received by the RTS, the image is passed through a (mixed-hardware/software) *Image Processing Pipeline* (IPP) implemented in PL (Fig. 5, green) and PS. After Gaussian filtering and Canny preprocessing via two IP cores in the PL, the remaining steps (edge tracking/detection, bounding box identification and distance look-up) are executed by a user-space application on the PS CPUs. Based on the present sphere position, the current sent through the electromagnet is adapted, which closes the control loop and enables levitation. To ensure stability, the overall processing latency has to be limited, which requires not only a mixed IPP but also *image acquisition* in PL.

With camera images arriving every 5.6 ms, a (PS) processing latency of already 3.8 ms and the software overhead of network stacks, traditional CPU-based image acquisition is insufficient. The VSS thus utilizes a dedicated image acquisition core in the PL that relies on a PS GigE controller for maximum efficiency. The *GigE Vision Bridge* (GEVB) core (Fig. 5, dark green) capitalizes on the controller's SG/DMA capability, and operates as follows. Instead of directly writing to the 256 receive buffers in DDR memory (cf. Sec. IV-A), the reconfigured controller is interacting with an alternate list of receive buffers that the GEVB (connected to the M-GP interface, ① in Fig. 5) provides. Once

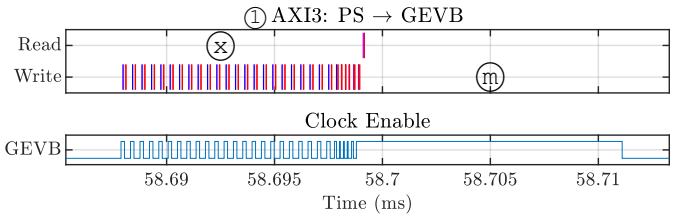

Fig. 7. Bus Transactions PS/GigE→GEVB and GEVB-CE for a GVSP Frame

the GigE controller has decoded one incoming Ethernet frame, it – like before – stores the payload at the given buffer location, which now, however, is under control of the GEVB. Internally, the GEVB thus includes two fabric-based memories, reachable via its AXI3 slave port (①). While one is the linear array with addresses and utilization status of four receive buffers (read by the GigE controller), the other contains the actual buffers (each large enough for one regular, full-sized Ethernet frame). As the former list points to the memory addresses of the latter buffers, incoming Ethernet frames are thus *redirected* from the external DDR memory into the PL. After reception, the GEVB analyzes each Ethernet frame to determine whether it holds some part of a camera image (i.e., is a valid GVSP packet), or if it is a non-image frame. While pixel data are extracted and sent to the IPP over the AXI4S port of the GEVB (③), non-image frames are *proxied* to the 256 CPU-controlled receive buffers in the DDR memory (Sec. IV-A) by means of the core's AXI3 master (②), which interfaces the DDR controller using one of the PS's four HP ports. Non-image frames thus reach the DDR memory as if regularly received without GEVB, which preserves subsequent IRQ- or polling-based processing by CPUs and network stacks.

Fig. 6 shows the bus transactions on all three AXI interfaces of the GEVB (①–③), over a period of two control iterations. Two clusters of 56 Ethernet frames (ⓒ/ⓐ) comprise a camera image each. Each cluster is preceded by one (additional) frame that transfers the GVSP Leader (pink), which indicate the start of a control iteration initiated by the camera. With reception of the first frame containing actual image pixels (blue), the GEVB initiates a new transaction on its AXI4S port (③ in Fig. 5), as indicated by the blue lines in Fig. 6 (③). Over the next 4.3 ms, the camera transmits the remaining image by means of 55 temporally distributed GVSP frames due to limited sensor readout.

With the IPP operating in-stream, i.e., without frame buffers, the hardware pipeline finishes shortly after the reception of the last GVSP frame, which ends the AXI4S transaction, as shown by the red lines in Fig. 6 (③). The AXI DMA core pushes the preprocessed camera image to the PS over ACP (⑥ in Fig. 5).

Fig. 6 also shows a non-image frame (ⓝ) that automatically is proxied back into the PS. Details of SG/DMA Proxying, and internals and characteristics of the IP cores can be found in [9].

C. Measurement Methodology: Voltage, Current & RTS States

We use a Xilinx ZC702 evaluation board in our experiments with the VSS and the proposed CG cores (Sec. V-C). Besides a medium-sized Zynq 7Z020 pSoC, it also features a GigE PHY, an external DDR memory of 1 GB, various other interfaces and peripherals, and a complex multi-rail supply solution – capable of measuring voltages and currents on each rail. The integrated

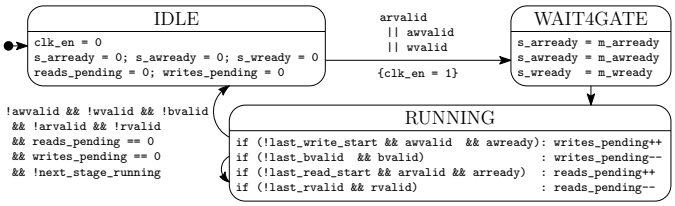

Fig. 8. AXI3 Clock Gating Controller: FSM for CE Generation (i.e., `clk_en`)

ADCs and readout logic of the supply controllers, however, are severely limited in terms of resolution, which renders them unsuitable for the evaluation of the high-speed VSS and our fine-granular CG methodology. Even if only a single rail is needed, the effective average sampling rate of below 3.5 kSPS – which yields less than 20 samples per VSS iteration of 5.6 ms – leaves a clear identification of intra-period savings out of reach. With our approach capable of such savings and, orthogonally, coarse ADC (current) readings in the tens of mA, the on-board supply solution cannot be used for accurate evaluation of our IP cores.

We thus deploy our custom external measurement system residing on an FPGA Mezzanine Card (FMC), which attaches to the ZC702 by means of analog voltage and current probes, and the digital FMC interface. With its 18-channel acquisition subsystem capturing analog 16-bit values at over 200 kilo-samples per second, this solution precisely identifies any changes in the energy consumption (due to, e.g., CG). An additional feature is the acquisition of *digital state information* from the RTS, such as (VSS) frame numbers and *CG enable/bypass* events (Fig. 1), enabling an automated analysis of achieved savings (Sec. V-C). Key concepts and detailed implementation information for this hybrid measurement/tracing methodology are available in [32].

V. Implementation and Experimental Evaluation

Based on the principles identified in Sec. III, we implement the two IPs for integration-level CG (Sec. V-A) to analyze achievable real-world energy savings and their resource consumption. With an accurate baseline (Sec. V-B), we quantify and discuss area and performance of the CG cores (Sec. V-C) for the VSS. Based thereon, Sec. V-D presents two regression-based savings models that enable extrapolations for other IPs and utilizations.

A. IP Cores for Integration-level Clock Gating

Despite not being tailored to any particular memory-mapped or stream-based bus interface, we use AXI3 and AXI4S (as the most widely used protocols in SoCs or FPGAs) to elaborate on the conceptual design (Sec. III) and particular implementation challenges of integration-level CG. Besides both generating the CE and gated clock signals, and manipulating certain incoming transactions, this also includes the detection of bus activity. We thus illustrate all this by means of the real-world VSS scenario (Sec. IV-B) for *AXI3* (Sec. V-A1) and *AXI4S CG* (Sec. V-A3).

1) The AXI3 CG Controller for Memory-mapped Interfaces: For an AXI3 interface, such as the connection between PS (i.e., its M-GP port) and GEVB shown in Fig. 5 (①), the intricacies of the underlying bus protocol have to be carefully considered. It comprises five independent channels – with two dedicated to addresses only [13]. Per transfer direction (i.e., read and write),

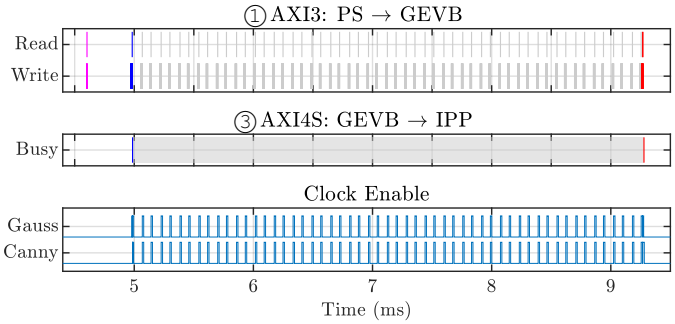

Fig. 9. Bus Transactions PS→GEVB→IPP and IPP-CEs in a Control Iteration

an address and a data channel each carry the requested memory location (32 bits), and a 32/64-bit word (GP vs. HP/ACP ports) per clock cycle. The remaining write response channel enables the (otherwise only receiving) slave to acknowledge successful writes. For increased performance, outstanding transactions are supported and have to be taken into account for the CG design.

Fig. 7 shows the transactions on the write and read channels between PS GigE Controller and GEVB (①) during reception of *one GVSP frame*. For write, 23 full-length bursts of 16 beats each are followed by a partial transfer. With each beat carrying 32 bits, all 1518 bytes of the GVSP frame thus are written to a receive buffer of the GEVB – followed by 5 short transactions to update metadata and tag the buffer as "used". Subsequently, the GigE Controller queries the GEVB for the next free buffer.

In combination with Fig. 6 that shows *two control iterations* (each containing 56 GVSP frames as detailed above), it can be clearly seen that the AXI3 interface (①), driving the GEVB, is mostly idle due to the slow camera sensor readout (Sec. IV-B). Thus, CG can be applied – as long as neither AXI transactions are pending (ⓧ in Fig. 7) nor the GEVB is moving data (ⓜ).

While the latter can easily be detected based on the GEVB's status, the former requires monitoring of all 5 AXI channels to ensure its timely wakeup. As each relies on a pair of handshake signals (`READY`/`VALID`) used to indicate not only transactions but also flow control, a *stateful CG controller* is needed, which detects the different types of transactions across all 5 channels.

Fig. 8 reproduces its main finite-state machine (FSM) generating the CE signal, while facing the following two challenges. Firstly, global/regional clock buffers (BUFxCE, cf. Sec. IV-A) *require one cycle* (after CE activation) until their clock outputs become active. Thus, incoming transactions have to be delayed until the downstream IP core is able to respond (i.e., it receives a clock). Otherwise, reads/writes from PS to GEVB will cause a timeout – rendering the VSS inoperative. The FSM overrides selected `READY` signals for one clock cycle – if CG is currently active – and thus (shortly) stalls incoming transactions until the downstream clock/core is active. This is implemented using the transition between the IDLE and WAIT4GATE states, which is taken once one of the three upstream-driven channels becomes active, as indicated by `arvalid`, `awvalid` and `wvalid`. At the same time, the BUFxCE is enabled (by means of `clk_en`). Once in WAIT4GATE, the `READY` signals are forwarded regularly and the FSM transitions to RUNNING after a clock cycle. In this state, handshake signals, `next_stage_running` and

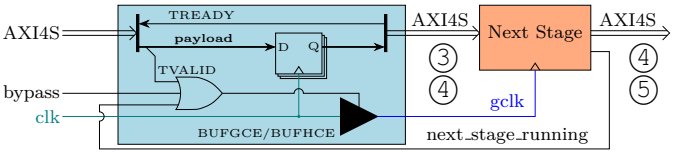

Fig. 10. AXI4S Clock Gating Controller for Gauss (③/④) and Canny (④/⑤)

two internal transaction counters are monitored, to detect if the clock can be disabled again by returning to the IDLE state. The counters track outstanding read and write transactions (that are currently delayed or processed by downstream interconnects or slaves, respectively), solving the second challenge. To achieve a resource-optimal implementation of each counter, we retrieve the configuration of the upstream interconnect (in our case via Xilinx' IP Integrator). Based on the defined maximum number of outstanding write and read transactions, the counters' widths are adjusted accordingly. As the AXI3 CG core is independent of address/data channels, its size depends on the counters only.

2) An application-specific CG Solution for both IPP Cores: As an intermediate step towards an integration-level CG for the AXI4S-only IPP (Sec. V-A3), we explored a – temporally and spatially – coarse-grained CG option that requires modification of the GEVB. The FSM pushing the video to the AXI4S port is extended to output an enable signal one clock cycle before the next AXI4S transaction (③ in Fig. 6/9) is initiated (cf. vertical blue lines). Based thereon, the gated clock for *both* IPP cores is enabled until Canny preprocessing ends (≈ red lines, Fig. 6/9).

3) The AXI4S CG Controller for Stream-based Interfaces: Given the protocol's simplicity, CG for any AXI4S-driven core (e.g., the two IPP ones) is considerably less complex. With one data payload and two handshake signals (`TREADY`/`TVALID` as per Fig. 10) only, the start of a single AXI4S transaction and its *individual* bus beats can easily be detected. Although the single transaction per control iteration (③ in Fig. 9) lasts 4.3 ms, the actual activity is considerably lower – as indicated by the space between GVSP frames (①). In contrast to Sec. V-A2, this enables a much more fine-grained CG – both *temporally* (as only actual bus activity, i.e., bus beats activate the CE) and *spatially* (as the IPP's two AXI4S-driven cores can be gated separately).

Not requiring internal states, the AXI4S CG controller has a less complex inner structure as given in Fig. 10. To compensate the BUFxCE activation delay, the core buffers the payload data in a register driven by the ungated clock (clk). As an alternative (in line with the above AXI3 CG controller), `TREADY` could be overridden. The clock output is enabled during transactions (as indicated by `TVALID`), in bypass mode, or if the (downstream) IPP core is running. As annotated in Fig. 10, this CG controller can be used for both AXI4S-driven cores in the IPP of the VSS (i.e., Gauss & Canny in Fig. 5) by simply instantiating it twice.

Like its AXI3 counterpart, the AXI4S CG controller features an optional `next_stage_running` input port to support any downstream cores with variable or even unknown tail latencies. This signal can be generated either by the particular core itself, as in case of the VSS (for GEVB, Gauss and Canny), or using an external downstream monitoring IP tailored to the scenario. As the AXI4S CG core buffers the payload data, its size varies.

TABLE I
FPGA RESOURCE USAGE: DEVICE VS. BASE IPS VS. VSS VS. CG CORES

Component / Subsys.	Slice LUTs	Slice Regs	Slices	BRAM[1]
Zynq 7Z020	53200	106400	13330	280
Base System				
Interconnect 1	1406	1394	567	0
Interconnect 2	415	489	172	0
State Tracing	462	484	197	4
Visual Servoing System				
GEVB	2745	2648	1185	16
Gauss	394	335	182	2
Canny	991	648	382	6
DMA	806	1158	401	2
Proposed Clock Gating IP Cores				
AXI3 (constant)	13	11	7	0
AXI4S (8 bit)[2]	1	11	7	0

NB: [1] 18 Kbit each / [2] resource usage of one 8-bit AXI4S CG core (instantiated twice)

B. Energy Baselines and Best-case Estimation

To characterize the achievable energy savings, the following baseline measurements were taken with all CG controllers disabled by means of their bypass signals. The PL needs 105 mW during idle, which increases to 113 mW once the camera starts to transmit. As the application is strictly periodic, power thus is proportional to energy and used instead. Stopping the entire PL by means of gating `FCLKx` from the PS yields a residual power consumption of 38 mW, saving 66% compared to active. Using the GEVB's PL utilization (36%) and activity (15.6%), a best-case reduction of $66\% \cdot 36\% \cdot (1\text{-}15.6\%) = 20.1\%$ is estimated.

Tab. I summarizes the fabric utilization of the design, which is dominated by AXI interconnects and the GEVB (particularly due to its own interconnect and BRAM-based receive buffers).

C. Optimized VSS: Savings and Observations

Being compatible to the traditional IP-based design flow, our AXI-based CG controllers are easily integrated into an existing mixed-PS/PL hardware pipeline like the VSS (Sec. IV-B). The *AXI3 CG controller* is implanted onto the link between PS and GEVB (① in Fig. 5), and manages the GEVB's clock. For the two IPP cores, a coarse-grained and our proposed per-IPP-core CG scenario is evaluated sequently. In the first, traditional case (cf. Sec. V-A2), all IPP cores are driven by a *shared clock* that is enabled during the entire AXI4S transaction. Secondly, each IPP core is individually managed by one *AXI4S controller* that is inserted upstream (at ③/④ for Gauss/Canny, cf. Fig. 5/10).

In both cases, an AXI3 core controls the GEVB. With two 2-bit counters, the entire CG core occupies 13 slice LUTs (SLs), 11 slice registers (SRs) and a BUFxCE, while generation logic of the GEVB's `next_stage_running` signal adds 1 SL and 1 SR. With our 8-bit payload, every AXI4S CG core uses 1 SL, 11 SRs and a BUFxCE, and IPP cores grow by 7 SLs and 1 SR.

As even the medium-sized Zynq 7Z020 provides 53200 SLs, 106400 SRs, 32 BUFGCEs and 72 BUFHCEs, the PL resource usage of CG cores and the added GEVB/IPP flags is negligible. As a point of comparison, the VSS pipeline alone (i.e., without surrounding infrastructure) already uses thousands of SLs/SRs, which is evident from Tab. I. Even for wide payload data (e.g.,

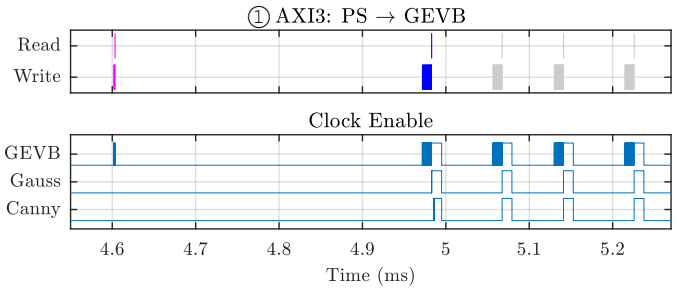

Fig. 11. Bus Transactions PS/GigE→GEVB and CEs for 1+4 Ethernet Frames

64 bit), the non-constant size of AXI4S CG controllers remains below that of any other IP core and their register stage even improves timing closure on the AXI4S links. For all experiments, the entire design is driven using a 125 MHz clock from the PS.

Fig. 7/11 show the resulting CE signal for the GEVB in time scales of one/multiple Ethernet frames, indicating a mostly disabled clock. In terms of power, savings of 21 mW or 18.3% are achieved for the GEVB with our AXI3 CG controller. For both IPP cores, the coarse-grained traditional approach only cuts the power by 1.3 mW due to the (near-)nonstop AXI4S transaction (③ in Fig. 6/9). In contrast, two of our AXI4S CG controllers reduce the power consumption of Gauss/Canny IPs by 2.6 mW (2.3%) and 5.7 mW (5%), respectively, i.e., over 6x more than for shared CG. As shown in Fig. 9/11, the generated CE signals for Gauss and Canny are as fine-grained as that for the GEVB, whereas the higher power reduction in case of the latter results from the IPP cores' lower resource utilization. In addition, actual GEVB savings are near the best-case estimate (Sec. V-B), which confirms its tail latency (ⓜ in Fig. 7) measured before. With total savings of approx. 29 mW (25.6%), our CG solution is as efficient as frequency reduction – while *not imposing any limits* on the maximum duty cycle or burstiness of transactions. As each of the three VSS IP cores (GEVB, Gauss, and Canny), like most others, requires at least hundreds of slices and several BRAMs, their fabric utilization nicely maps to the size of CRs in current FPGAs (Sec. IV-A). With over ten CG-capable clock signals in every CR, the proposed integration- or core-level CG methodology thus is neither too fine- nor too coarse-granular in relation to now prevailing medium-grained clocking resources. Depending on the core size and available resources, the system designer can easily migrate between global and regional clocks using a configuration parameter of our cores. Correct operation was verified for both a BUFGCE- and BUFHCE-gated version.

It should be noted that all the aforementioned savings apply to regular operation of the VSS, i.e., the camera capturing at its upper limit of 178 frames per second. In this configuration, the control goal of stable levitation can be reached with PID or LQ controllers and appropriate gains. Higher sampling/frame rates would improve stability, but are impossible due to the camera's sensor. As *lower utilizations* like the 10 frames per second case in Fig. 1 (with overall savings of 31.7%, compared to 25.6% at full speed) generally yield *higher savings* due to less switching activity, a wider range of rates has to be explored methodically.

To this end, we recorded the GVSP frames sent by the actual camera ("Mako") to create a virtual counterpart ("vCam") with higher maximum frame rate. By speeding up the stream replay, we are able to reach the saturation limits of the VSS at 300 and 800 frames per second for software and hardware, respectively. With frame numbers and CG configuration captured along with the analog values (Sec. IV-C), actual frame rate, active power, and relative savings are automatically measured. Tab. II shows an excerpt of the captured data including the worst case of over 800 frames per second. Despite such a high utilization, our CG controllers still yield savings of 10.3% (at negligible overheads w.r.t. both the added pipeline latency of 8 ns and PL resources).

TABLE II
CLOCK GATING FOR VSS (EXCERPT): TEST SETUP, POWER AND SAVINGS

Frame Rate Set	Frame Rate Real	Video source	Power active	Relative Savings GEVB	Relative Savings Gauss	Relative Savings Canny
10	10.0	Mako	106.1 mW	22.9%	2.8%	6.1%
50	50.0	Mako	107.6 mW	21.7%	2.8%	5.9%
100	100.0	Mako	109.7 mW	20.3%	2.6%	5.5%
150	150.0	Mako	111.5 mW	19.0%	2.3%	5.2%
max	178.1	Mako	112.7 mW	18.3%	2.3%	5.0%
250	249.9	vCam	115.8 mW	17.3%	2.1%	4.7%
400	395.7	vCam	121.1 mW	14.3%	1.8%	3.9%
550	539.4	vCam	126.9 mW	11.6%	1.4%	3.2%
700	675.4	vCam	132.5 mW	9.1%	1.3%	2.7%
850	805.3	vCam	137.8 mW	7.0%	1.1%	2.2%

D. Savings Model for Integration-level CG

As the achievable savings for CG-based energy optimization heavily depend on core size and activity in the design (e.g., the VSS), it can be hard to estimate them in advance and for other, not-yet-evaluated scenarios. Given the known, linear relation between activity and power, we, however, argue that – for one given type of FPGA fabric – such extrapolations might indeed be feasible based on our exhaustive measurements. To this end, we present and investigate two models for achievable savings, each populated via an ordinary least squares regression. While Model 1 (M1) is strictly linear, Model 2 (M2) also contains an interaction term. Both map the independent variables of IP size (C) and activity (A) to the dependent variable Δ of the relative savings, cf. (1). In M1, no interaction is assumed, thus $\gamma = 0$.

$$\Delta(A, C, \delta, \alpha, \beta, \gamma) = \delta + A \cdot \alpha + C \cdot \beta + A \cdot C \cdot \gamma \qquad (1)$$

In R, we identify δ, α, β, and γ (M2 only) by minimizing each total quadratic error between our measurements and the model. Due to our 108 observations (Sec. V-C), both models are well defined, although M1 only yields $R^2 = 0.931$, i.e., it is able to explain 93% of the measured savings values. M2, on the other hand, results in a near-perfect coefficient of determination with $R^2 = 0.998$ and thus nicely predicts the potential savings Δ as a function of A and C – with δ, α, β, and γ as given in (2).

$$\delta = -1.1277 \;;\; \alpha = 0.0013 \;;\; \beta = 0.0198 \;;\; \gamma = -1.784 \cdot 10^{-5} \qquad (2)$$

Fig. 12 shows the actual measurements (single dots, cf. Tab. II) and the predicted savings based on M2. It can be clearly seen that the estimation errors are low, as already indicated by the large R^2 of M2. As our measurements cover three separate IP cores, two bus interfaces, and dozens of activity values (i.e., frame rates), we are confident that M2 is suitable to predict the *potential savings* of our solution for various scenarios and IPs.

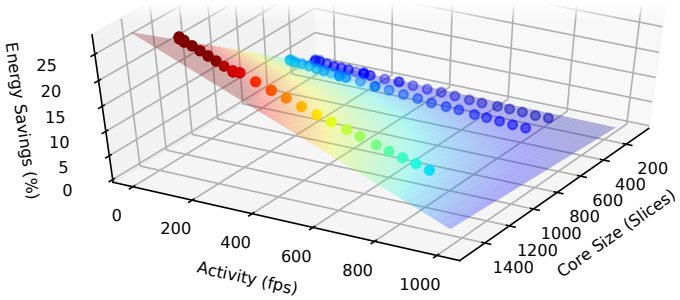

Fig. 12. Energy Savings: Measured (dots) and Predicted via M2 (surface plot)

VI. Conclusion

In this paper, we evaluated clock gating on integration level for today's IP-driven FPGA/pSoC pipelines to reduce the dynamic energy consumption of the underlying device fabric. Using the fact that the majority of current FPGA-based real-time systems are designed using IP cores and standardized bus interfaces, we presented two efficient AXI-based clock gating controllers that neither require new design flows nor negatively impact temporal system characteristics. Based on application information on the bus, downstream cores are accurately optimized for energy.

Our experimental evaluation of a real-world VSS application on a Zynq pSoC shows overall energy savings of 25.6%, while maintaining both timing and I/O via the SoC part of the device. For future work, we intend to extend our optimizations beyond fabrics by devising, e.g., delay-aware DFS strategies for CPUs.

Acknowledgment

The authors would like to thank the anonymous reviewer(s) for pointing out the related work in reference [25]. While our work implements a similar concept in the context of FPGAs and on a per-core basis, the comparison made this paper more complete.

References

[1] M. B. Srivastava, A. P. Chandrakasan, and R. W. Brodersen, "Predictive system shutdown and other architectural techniques for energy efficient programmable computation," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, vol. 4, no. 1, pp. 42–55, Mar. 1996.

[2] A. W. Min, R. Wang, J. Tsai, and T. C. Tai, "Joint optimization of dvfs and low-power sleep-state selection for mobile platforms," in *2014 IEEE International Conference on Communications (ICC)*.

[3] I. Bate, J. McDermid, and P. Nightingale, "Establishing timing requirements for control loops in real-time systems," *Microprocessors and Microsystems*, vol. 27, no. 4, pp. 159–169, May 2003.

[4] X. Mei, X. Chu, H. Liu, Y. Leung, and Z. Li, "Energy efficient real-time task scheduling on cpu-gpu hybrid clusters," in *2017 36th IEEE Conference on Computer Communications (INFOCOM)*.

[5] M. Hosseinabady and J. L. Nunez-Yanez, "Energy optimization of fpga-based stream-oriented computing with power gating," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*.

[6] M. Törngren, "Fundamentals of implementing real-time control applications in distributed computer systems," *Real-Time Systems*, vol. 14, no. 3, pp. 219–250, May 1998.

[7] T. Kim, H. Shin, and N. Chang, "Deadline assignment to reduce output jitter of real-time tasks," in *2000 16th IFAC Workshop on Distributed Computer Control Systems (DCCS)*.

[8] G. Buttazzo and A. Cervin, "Comparative assessment and evaluation of jitter control methods," in *2007 15th Conference on Real-Time and Network Systems (RTNS)*.

[9] M. Geier, F. Pitzl, and S. Chakraborty, "GigE vision data acquisition for visual servoing using sg/dma proxying," in *2016 14th ACM/IEEE Symposium on Embedded Systems for Real-Time Multimedia (ESTIMedia)*.

[10] C. Claus, R. Ahmed, F. Altenried, and W. Stechele, "Towards rapid dynamic partial reconfiguration in video-based driver assistance systems," in *2010 6th International Symposium on Applied Reconfigurable Computing: Architectures, Tools and Applications (ARC)*.

[11] M. Hosseinabady and J. L. Nunez-Yanez, "Dynamic energy management of fpga accelerators in embedded systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 3, pp. 63:1–63:26, May 2018.

[12] Xilinx Inc., "Vivado Design Suite User Guide: Designing with IP," https://www.xilinx.com/cgi-bin/docs/rdoc?v=2019_2;d=ug896-vivado-ip.pdf, 2020, [UG896; Revision v2019.2 of March 3, 2020].

[13] ARM Ltd., "AMBA Specifications," https://www.arm.com/products/silicon-ip-system/embedded-system-design/amba-specifications, 2019.

[14] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 26, no. 2, pp. 203–215, Feb. 2007.

[15] Xilinx Inc., "7 Series FPGAs Memory Resources," https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf, 2019, [UG473; Revision 1.14 of July 3, 2019].

[16] Xilinx Inc., "7 Series FPGAs Clocking Resources," https://www.xilinx.com/support/documentation/user_guides/ug472_7Series_Clocking.pdf, 2018, [UG472; Revision 1.14 of July 30, 2018].

[17] Xilinx Inc., "Reducing Switching Power with Intelligent Clock Gating," https://www.xilinx.com/support/documentation/white_papers/wp370_Intelligent_Clock_Gating.pdf, 2013, [WP370; Revision 1.4 of August 29, 2013].

[18] A. Becher, J. Pirkl, A. Herrmann, J. Teich, and S. Wildermann, "Hybrid energy-aware reconfiguration management on xilinx zynq socs," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*.

[19] M. Hosseinabady and J. L. Nunez-Yanez, "Run-time power gating in hybrid arm-fpga devices," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*.

[20] X. Chang, M. Zhang, G. Zhang, Z. Zhang, and J. Wang, "Adaptive clock gating technique for low power ip core in soc design," in *2007 IEEE International Symposium on Circuits and Systems (ISCAS)*.

[21] A. Hermanek, M. Kunes, and M. Tichy, "Reducing power consumption of an embedded dsp platform through the clock-gating technique," in *2010 20th International Conference on Field Programmable Logic and Applications (FPL)*.

[22] M. R. Alam, M. E. S. Nasab, and S. M. Fakhraie, "Power efficient high-level synthesis by centralized and fine-grained clock gating," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 34, no. 12, pp. 1954–1963, Dec. 2015.

[23] E. Bezati, S. Casale-Brunet, M. Mattavelli, and J. W. Janneck, "Clock-gating of streaming applications for energy efficient implementations on fpgas," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 4, pp. 699–703, Apr. 2017.

[24] C. A. Khatib, C. Aupetit, A. Chagoya, C. Chevalier, G. Sicard, and L. Fesquet, "Distributed asynchronous controllers for clock management in low power systems," in *2014 21st IEEE International Conference on Electronics, Circuits and Systems (ICECS)*.

[25] J. Yun, L. Liao, and B. Jeong, "Bus system in soc and method of gating root clocks therefor," U.S. Patent 9 152 213, Oct. 6, 2015.

[26] Xilinx Inc., "7 Series FPGAs Configurable Logic Block," https://www.xilinx.com/support/documentation/user_guides/ug474_7Series_CLB.pdf, 2016, [UG474; Revision 1.8 of September 27, 2016].

[27] Y. Zhang, J. Roivainen, and A. Mammela, "Clock-gating in fpgas: A novel and comparative evaluation," in *2006 9th EUROMICRO Conference on Digital System Design (DSD)*.

[28] M. Athans, "The role and use of the stochastic linear-quadratic-gaussian problem in control system design," *IEEE Transactions on Automatic Control*, vol. 16, no. 6, pp. 529–552, Dec. 1971.

[29] B. Wittenmark, "Integrators, nonlinearities, and anti-reset windup for different control structures," in *1989 American Control Conference*.

[30] Xilinx Inc., "Zynq-7000 All Programmable SoC Technical Reference Manual," https://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf, 2018, [UG585; Revision 1.12.2 of July 1, 2018].

[31] AIA, "GigE Vision Main Page - AIA Vision Standards," https://www.visiononline.org/vision-standards-details.cfm?type=5, 2018.

[32] M. Geier, M. Brändle, D. Faller, and S. Chakraborty, "Debugging fpga-accelerated real-time systems," in *2020 25th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.