Debugging FPGA-accelerated Real-time Systems

Martin Geier, Marian Brändle, Dominik Faller and Samarjit Chakraborty
Chair of Real-Time Computer Systems
Technical University of Munich
geier/braendle/faller/chakraborty@rcs.ei.tum.de

***Abstract*—The high computation/communication requirements along with reliability needs and limited power budgets necessitate complex processing platforms for emerging autonomous systems. Due to the current focus on performance, however, such platforms are increasingly difficult to predict and analyze. This holds true in terms of both performance (e.g., behavioral and temporal) aspects and power consumption. Ensuring functional safety thus requires new techniques to analyze performance, predictability and power. In this paper, we thus propose a novel hybrid tracing methodology to monitor (and, subsequently, optimize) temporal, functional and energy-related properties of Real-time Systems (RTSs). We target current Programmable SoCs (pSoCs) integrating a fixed-function System-on-Chip (SoC) with flexible Field Programmable Gate Array (FPGA) fabric. Although such heterogeneous systems are well suited for high-end, mixed-hardware/software real-time pipelines, they also offer more complex performance/energy trade-offs than software-only platforms. To systematically exploit this complexity, we present a resource-efficient trace IP core for the pSoC's fabric and an external measurement/interface system – jointly capturing hybrid power/state traces for subsequent (i.e., offline) analysis. By fusing state data from our IP core with events-of-interest gathered from power traces of pSoC and co-monitored I/O components, we gain a holistic view on temporal RTS aspects. Events and synchronized multi-rail power data jointly extend the debugging coverage via automated identification of processing phases, computation of energy baselines, and estimation of potential savings. Our solution thus integrates functional, temporal and energy monitoring into a single, unified workflow, which, in contrast to traditional separate tools, delivers valuable new insights helpful during debugging and reduces both cost and effort. Experimental evaluations on a Zynq-based Visual Servoing System show the method's various benefits.**

I. Introduction

With many embedded systems today operating in time-critical application domains at highest performance, the required complex processing platforms are increasingly hard to predict w.r.t. their temporal behavior. This is particularly challenging for hardware-accelerated Real-time Systems (RTSs) that – in addition to already complex application software – rely on custom hardware pipelines to reach the performance goals. Commonly, such hardware is implemented via a Field Programmable Gate Array (FPGA) [1], [2] – yielding a heterogeneous RTS, which, e.g., then closes the loop in networked control systems (Fig. 1).

In such systems, it is crucial to know the *overall processing latency* from arrival of sensor inputs to the transmission of an actuation signal to ensure *control stability*. In many application scenarios and, particularly, when using heterogeneous architectures, however, these latencies are not only hard to analyze, but also adversely affected by most *power management* techniques required to keep the RTS within supply and cooling limitations. To systematically gain insight on the inherent trade-off between energy and latencies, we argue that both have to be *jointly* dealt with – which exceeds the coverage of traditional tools handling *either* temporal-functional properties *or* energy (Sec. II-C). We thus propose a hybrid debugging methodology to capitalize on our insights that latency and power debugging should be a joint process and, without loss of coverage, be performed at a *higher level of abstraction* – compared to traditional FPGA tools (like, e.g., Intel SignalTap) as detailed in Sec. II-D on page 5 (top r.). As further conceptual contribution, we make a case for precise, *per-component energy monitoring* that includes control-related I/O devices (such as network peripherals) to capture the RTS's total processing latency – from data input to control output. To support our insights, we present an implementation strategy for such tracing and demonstrate its benefits on a networked RTS.

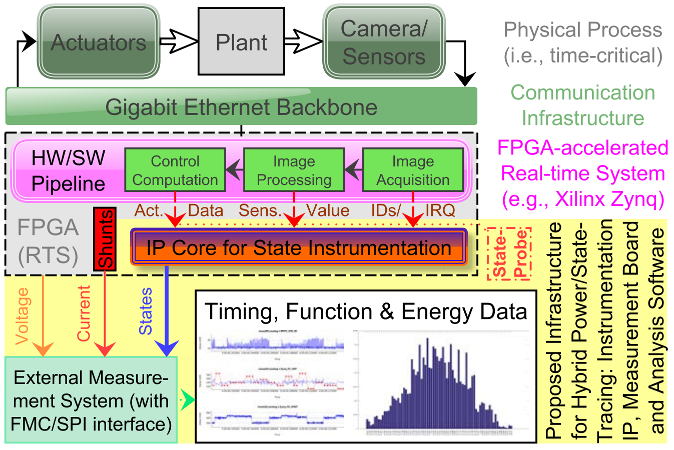
Joint Monitoring: As it turns out, such a hybrid method has significant advantages over existing solutions. Even though the proposed level of abstraction implements the crucial functional monitoring solely via (application) macro-phases, the resulting traces cover the following major behavioral aspects of the RTS, with – due to our lightweight instrumentation approach – *near-zero impact* on its resource usage and thus function, timing and energy consumption. Firstly, we are able to capture *partial* and, most importantly for heterogeneous RTSs (using FPGA-based, e.g., sensor acquisition and/or processing), *true input-to-output latencies*. Secondly, our methodology combines the automated identification of application phases with multi-rail energy data, enabling the computation of *per-phase, per-component energy baselines* and an *estimation of best-case energy savings*. Based thereon, the *optimization of RTS latencies and energy* becomes a significantly less complex process (with shorter turnarounds).

Although relatively new for FPGA-based RTSs, the underlying challenge of *balancing performance vs. energy* has steadily been intensified by the developments in computer architecture, with existing tools persistently lagging behind even for timing-only (i.e., not energy-related) analysis problems [3]. Balancing latencies and energy of mixed-hardware/software applications, however, causes additional intricacies for tools and developers, as such particular, heterogeneous platforms offer an even larger design space than traditional, software-only RTSs. Whilst, e.g., FPGA-accelerated real-time pipelines may deliver the required performance, their temporal behavior is particularly difficult to predict and measure – due to the complex interactions between software and partially application-specific hardware layers [4]. Whilst *software-based tools*, for instance, are unable to capture processing latencies caused by such custom hardware, existing Intellectual Property (*IP*) *cores for FPGA debugging* are costly in terms of resource usage and limited w.r.t. their coverage (as shown in Sec. II-B) – with the RTS's end-to-end latencies thus also remaining way out of reach. In addition, those functional-temporal tools are totally separate from *energy monitoring*, and vice versa. Thus, neither energy baselines per processing phase nor performance/energy trade-offs can be made available to the system designer. Integrating both classes of tools into a unified workflow is a non-trivial task – not only due to their individual limitations, but also because they, e.g., rely on different notions of time and often are black-box IP or proprietary measurement solutions (cf. Sec. II-C). In summary, the continuously increasing complexity of current FPGA-accelerated RTSs (that results from tight requirements) already heavily interferes with energy or latency analysis, thus necessitating new, less restricted tools.

Proposed Methodology: In this paper, we introduce *Hybrid Power/State-Tracing* as a novel debugging methodology using *synchronized power/state data* for the analysis of FPGA-based (heterogeneous) RTSs. By combining a resource-efficient *trace IP core* instantiated in the RTS's FPGA fabric with an External Measurement Subsystem (EMS), we are able to synchronously capture comprehensive *state information* plus *energy data* of at least nine supply rails. Whilst the IP core (Fig. 1, orange+blue) flexibly aggregates various system states using software instrumentation and/or by tapping relevant hardware signals (dashed red), its data stream (blue) together with voltage/current probes (orange/red) enable our EMS (bottom left) to generate a hybrid power/state trace (Fig. 1, white box). We also extend the power monitoring to crucial I/O components (such as Ethernet PHYs) and fuse the resulting timestamps with those extracted from the state trace to simultaneously gain a holistic view on the RTS's temporal behavior – including its end-to-end latencies. As our EMS (see also Fig. 2) adds a precise, sub-µs timestamp to each data point, both *temporal* events-of-interest (from I/O monitoring and state trace) and additional *functional* information (also contained in the trace) can easily be aligned to the power data. This enables a Python-based analysis tool to compute precise *energy* figures – per processing phase and RTS subcomponent. Overall, the proposed hybrid power/state trace thus carries all information required to perform unified temporal, functional and energy monitoring of an FPGA-based RTS with low effort.

We present flexible implementation options for the proposed trace IP core that differ in supported features and range from FPGA-only monitoring to capturing the entire RTS – enabling not only processor tracing using *code instrumentation*, but also later, *automated* low-overhead trace readback. Our RTS-under-test is a Xilinx Zynq that combines a fixed-function, dual-core Processing System (PS) with FPGA-equivalent Programmable Logic (PL) fabric, residing on a ZC702 development board. Its FPGA Mezzanine Card (FMC) interface enables us to connect our EMS that not only acquires *RTS power/states* but also integrates two *co-monitored* Gigabit Ethernet (GigE) PHYs as networked RTSs often rely on GigE to interface the outside world.

Applications and Benefits: Together, our proposed trace IP core and custom FMC-sized measurement system implement a *cost- and resource-efficient solution* for unified monitoring of a heterogeneous RTS – reaching beyond 200 kSPS (kilo-samples per second) for voltage or current values, and over 25 kSPS for state data. We illustrate the various *applications and benefits of unified power/state monitoring* for a PL workload with highly variable power demand and a real-world mixed-hardware/software Visual Servoing System (VSS) that implements a closed control loop – driven by a high-rate video stream from a GigE Vision camera (Fig. 1). In case of the VSS, it is crucial to know the RTS's total latency to *ensure stability* in control design. A key question thus is *How long does it take from arrival of one, entire camera image to transmission of an actuation value*. The hybrid power/state trace in Fig. 8 fully describes an iteration of the VSS. With incoming Ethernet frames (green) automatically identified on a PHY supply rail (gray, magnified for visibility) and fused to various state values gathered in PS and PL (red), the RTS's temporal behavior is accurately captured – sufficient to answer the above and further, crucial questions (cf. Sec. VI). In summary, the main contributions of this paper are



Actuators
Plant
Camera/ Sensors
Physical Process (i.e., time-critical)
Gigabit Ethernet Backbone
Communication Infrastructure
HW/SW Pipeline
Control Computation
Image Processing
Image Acquisition
FPGA-accelerated Real-time System (e.g., Xilinx Zynq)
Act. Data
Sens. Value
IDs/ IRQ
FPGA (RTS)
Shunts
IP Core for State Instrumentation
State-Probe
Voltage
Current
States
Timing, Function & Energy Data
External Measurement System (with FMC/SPI interface)
Proposed Infrastructure for Hybrid Power/State-Tracing: Instrumentation IP, Measurement Board and Analysis Software

Fig. 1. Control loop (top) closed by heterogeneous RTS (center), instrumented via our IP Core of the proposed Infrastructure for Power/State Tracing (bottom)

- *Hybrid Tracing* as novel methodology for *unified temporal, functional & energy monitoring* of FPGA-accelerated RTSs, using power consumption and various state signals,
- Automated *Identification of I/O and Processing Phases*, per-phase/component *Energy Baselines*, estimated *Best-Case Energy Savings* and holistic *Latency Monitoring*,
- Flexible implementation options for the resource-efficient *trace IP core* extracting the state trace from PS/PL and
- *Low-overhead Trace Readback* for intervention-free tests.

Outline: The rest of this paper is organized as follows. Technical background and related work are presented in Sec. II. Then, Sec. III introduces key concepts of the proposed methodology, followed by Sec. IV detailing a hardware/software architecture to implement such hybrid tracing – using our resource-efficient IP core (Sec. IV-A) with its various implementation and PS/PL interface options (Sec. IV-B/C/D). Next, Sec. V describes EMS (Sec. V-A) and two reference workloads – a synthetic memory test and a real-world VSS (Sec. V-B/C). Based thereon, Sec. VI shows the various *applications and benefits* of hybrid traces for unified temporal, functional and energy monitoring on a Zynq-based RTS/platform. We finally conclude our work in Sec. VII. The appendices A and B supplement quantitative comparisons.

II. Background and Related Work

Over the last decades, continuous advances in chip technology, design tools and resulting hardware architectures sustained the deployment of increasingly complex Systems-on-Chip (SoCs). Besides intensifying the various cost-related challenges faced during design, test and integration (such as, e.g., development, fabrication and verification efforts), the resulting computation and communication densities also increase the *energy demand* of high-performance processing platforms. In many domains, this restricts the choice of platform depending on both supply and cooling constraints of the application scenario at hand. In case of a lot of emerging autonomous systems, for instance, no particular architecture has yet been identified as being the most efficient. Instead, each manufacturer of (partially) autonomous vehicles relies on a unique combination of processing elements including Central Processing Units (CPUs), Graphics Processing Units (GPUs), Digital Signal Processors (DSPs) and, more recently, dedicated Deep Neural Network (DNN) accelerators, each increasing power consumption and heat dissipation of the entire RTS – or Application-specific Integrated Circuit (ASIC).

At the same time, however, various optimization strategies for fixed-function and, in the last years, FPGA devices assist in reducing the overall energy demand. In case of fixed-function CPUs, DSPs or, generally, ASICs, this includes methods such as clock and/or power gating, Dynamic Voltage and Frequency Scaling (DVFS) and incremental processor sleep states – controlled by hardware, firmware or software [5]–[9]. Although traditionally lagging behind in terms of energy efficiency [10], FPGA-based systems now implement not only selected methods well known from ASIC design (e.g., glitch-free clock gates and/or multiplexers), but also features specific to the particular reconfigurable FPGA fabric including automatic power gating of uninstantiated memory resources [11]. Together with several architecture primitives for both clock generation (e.g., clock management tiles) and distribution (e.g., global, regional or I/O clock trees) [12], current FPGAs thus support an increasing number of *fine-grained power management* features [13]. This enables both traditional – i.e., fabric-only – devices and, more recently, heterogeneous *Programmable SoCs (pSoCs)* to reach more energy-sensitive application domains such as, e.g., automotive and robotics. By combining a fixed-function (and thus power-efficient) multi-core SoC with a flexible, reconfigurable FPGA fabric, such heterogeneous platforms are well suited for mixed-hardware/software processing pipelines [2]. In addition, the Dynamic Partial Reconfiguration (DPR) capability of many devices [1] also enables energy-aware runtime strategies [14].

A. The Challenge: Power Management vs. System Latencies

Above optimization strategies, however, come at a cost. This holds true for design time – due to, e.g., increased development and verification efforts – and also during manufacturing (where costly advances in process scaling, e.g., influence the static and dynamic power consumption). It, in addition, applies to regular system operation, as many energy saving techniques also affect the *temporal behavior* of the system. This trade-off is of particular importance for (mainly hard) RTSs – as they operate under predefined *latency constraints* in safety-critical environments, where neither a deadline violation (caused by longer response times) nor unforeseen jitter can be tolerated [15]. For a *closed-loop control system* (Fig. 1), for instance, the overall sensor-to-actuator latency has to be limited to avoid physical oscillations and instabilities. It is heavily influenced by the communication and computation delays caused by both network (in case of a distributed setup) and the various processing steps on the RTS.

To ensure compliance with such timing constraints and support developers during system design and debugging, *formal methods* and, at the other end of the spectrum, *measurement-based approaches* are widely used. Whilst the former enable various latency-aware online strategies for system-level power management [16], [17], individual components such as processors [18] plus interconnects for SoCs [19] and datacenters [20] are commonly co-optimized (w.r.t. energy, latency and area) at design time. Dealing with the large variations in (i.e., software) execution times caused by, e.g., pipelining or shared hardware resources, however, remains a challenging task – even without any notion of energy [3]. Although the formal methods such as (static) timing analysis yield safe, but often rather pessimistic *upper timing bounds*, they are practically infeasible due to the extremely large state space of modern hardware and software architectures [21], [22]. The measurement-based approaches, on the other hand, are not limited to scenarios with restricted complexity, but – inherently – only capture weak *maximum observed* execution times that may or may not be exceeded in the future. Still, it is common practice in safety-critical industries such as automotive and avionics to perform elaborate *latency measurements* over extended periods [23], as the resulting high watermarks (i.e., the longest observed delays) approximate the actual worst-case latencies over continuous system operation. Similar challenges exist on GPUs, where recent modeling [24], analysis [25] and lock-stepping [26] efforts aim at making their massive parallelism available for safety-critical environments.

In addition to such primarily latency-focused measurement approaches, various *functional debugging utilities* also provide helpful insights into the temporal behavior of the device-under-test (DUT). This holds true for not only the extensive – event-based – runtime tracing facilities provided by current real-time operating systems [23], but also various (internal and external) solutions for functional debugging or tracing. Relying on either standardized interfaces (e.g., JTAG and NEXUS with its high-speed auxiliary port) or proprietary implementations (such as ARM's CoreSight), they enable both functional debugging (by, e.g., code/data breakpoints or memory inspection) and mixed-functional/temporal monitoring of current CPUs [27], [28]. In contrast to the above software-driven methods (relying on code instrumentation), the DUT remains unchanged during analysis. Although the resulting traces accurately capture a combination of functional and temporal CPU behavior for offline or online analysis [28], their high data rates (and large file sizes) render this approach rather costly. In addition, they – like the other monitoring techniques – lack information regarding the power consumption and thus cannot be used (alone) to balance energy versus processing latencies during the optimization of an RTS.

B. Traditional Techniques for RTS Power and/or Latencies

Besides recent work on large-scale and (at the moment) thus inherently soft real-time datacenter workloads [20], [29]–[31], modeling, management and optimization techniques for RTS power and latency are extensively studied – both together and separately. As *static leakage currents* (increasingly) dominate the overall consumption due to shrinking process geometries, only optimizing for *dynamic power* by means of, e.g., D(V)FS is no longer sufficient for energy-constrained RTSs [9]. More recent techniques thus additionally use the flexible sleep states (implementing fine-grained clock and power gating) of current CPUs to optimally schedule task sets described by traditional periodic models [9], [32] or, generally, irregular event patterns captured by arrival curves known from real-time calculus [33].

Power modeling itself not only depends on various characteristics of the underlying process technology, but (in addition to and in line with latency modeling) also requires some notion of circuit activities to estimate the resulting demand [10]. For FPGA and pSoC power, the available tools range from spreadsheets (relying on precise user-supplied design characteristics, e.g., Xilinx' Power Estimator [34]) to model-based workflows, which use activity traces from prior functional simulation [35].

Power Monitoring: During measurements (contrary to simulation and modeling approaches) on actual devices including, in particular, FPGA-based platforms, however, an RTS' power consumption and its various latencies are commonly monitored separately. Although most of today's FPGAs do not yet support Dynamic Voltage Scaling (DVS), accurate power measurement requires sensing the near-constant voltages and highly dynamic currents on all the supply rails of interest. In case of a pSoC, its individual subcomponents both in the fixed-function SoC (such as CPUs, memory controllers and I/Os) and in the configurable fabric (e.g., logic slices, BlockRAM, phase-locked loops or I/O blocks) require independent supplies. Comprehensive energy monitoring – suitable for a multitude of system designs (each exhibiting a unique distribution of demand) – thus might have to cover up to ten rails. As current is generally measured either using a series resistor (i.e., a shunt) or an external, monitored source (if – at all – electrically feasible), per-rail measurements depend on the (immutable) shunt resistors that are part of many systems' on-board supply solutions [14], [36], [37]. In case the particular device features analog inputs (using, e.g., XADC on Zynq [38]), supply and amplified shunt voltages can be filtered (to avoid aliasing), converted and stored internally. Without access to shunts, only the overall power consumption of the RTS can be captured on its external supply side, which complicates identification of per-component activities. Efficiency variations of the on-board DC/DC converters caused by load changes also have a significant impact on measurement accuracy [39]–[41].

Latency monitoring, on the other hand, is mostly realized using *software instrumentation* or by the addition of dedicated *measurement hardware*. Although the former requires changes to the DUT's software and thus itself may affect the temporal behavior, it enables in situ monitoring out in the field. The latter, by contrast, is capable of capturing latency figures without any DUT modification, but depends on (potentially expensive, external) monitoring equipment that is unavailable during regular system operation – due to cost, space and/or practicability limitations. Above instrumentation-based approaches are thus often applied both during development of an individual system, and in case of large-scale, long-term monitoring after rollout in the field. High watermarks (which capture the longest observed latencies) are extensively used in safety-critical domains such as automotive and avionics [42]. Furthermore, they would even drive (or improve) probabilistic timing analysis [22]. Real-time operating systems (e.g., Linux with PREEMPT_RT patch [43]) also integrate extensive tracing facilities to analyze the kernel's task scheduler, the applications' executions times and various other latency-relevant statistics during system operation [23]. Whilst many CPUs include dedicated monitoring blocks (e.g., Intel's performance monitoring units [44]) used by above and proprietary analysis tools (such as VTune Amplifier [45]), they capture latency values on a per-cycle level, e.g., for single CPU events such as memory accesses with cache miss – far too fine-grained for end-to-end latency monitoring. In GPU-accelerated architectures, both the processing and communication latencies generally are neither well documented nor easily quantified as vendor-supplied drivers, tools and device visibility are limited.

Traditional FPGA Debug/Trace Options: Current pSoCs, however, can integrate above external tracing/monitoring hardware to a certain degree. This holds true for both combined debugging/trace acquisition solutions available for current CPUs with appropriate high-speed interfaces (e.g., ARM's CoreSight ETM ports and MIPS OCI) [28] and dedicated external latency monitoring hardware [46]. Whilst integration of such monitors is a relatively new development [4], versatile debuggers/tracers for FPGAs and pSoCs are well established due to their fabric's short reconfiguration times. Its logic/storage enables temporary (i.e., debug-only) and flexible (e.g., w.r.t. their signal coverage) instantiation of complex *Embedded Logic Analyzers* (ELAs). Such (mixed-hardware/software) solutions are readily included in the development tools of all major FPGA vendors and combine a configurable, yet proprietary IP core for data acquisition with software facilities for subsequent offline analysis. Xilinx' Integrated Logic Analyzer (ILA) [47] or Intel's SignalTap [48], for instance, are fabric-based digital logic analyzers containing comparator/trigger circuitry (via logic resources) and a storage buffer in, e.g., BlockRAM (BRAM). As long as required logic and memory resources are available, these solutions provide a relatively fast way to gather cycle-accurate waveforms of a few manually selected internal signals. The instantiation of such IP cores, however, affects both logic placement and signal routing on the fabric, and thus alters the design [49], [50] – particularly in situations with tight and/or asynchronous timing constraints. Their achievable capture lengths and depths are limited by the availability of internal memory resources, which often already are occupied by the actual application's IPs. In contrast to the debug/trace facilities integrated in most fixed-function devices (such as CPUs or DSPs), it is, however, possible to remove ELA-based solutions from the design once no longer required and reassign the now free FPGA resources. The selection of all capture signals traditionally is an iterative, manual and (due to long synthesis/implementation runtimes) also time-consuming process. Recent research thus proposes helpful extensions, e.g., automated identification of the relevant signals for both hand-crafted designs [51], [52] and, more recently, those generated from high-level synthesis tools [53]–[56], also including multi-threaded circuits [57]. Further, various techniques to simplify a post-implementation trace insertion and signal selection enable flexible tracing without long design iterations using bitstream manipulation [49], DPR [58], [59] or, recently, even both [60].

C. Limitations of Traditional Approaches and Synchronization

Although ELAs provide cycle-accurate waveforms at design speeds (e.g., several hundreds of MHz) and fine-grained trigger conditions (suitable for functional and temporal monitoring on, e.g., register-level), the resulting traces still lack crucial system aspects. They neither cover *extended intervals* (such as dozens of milliseconds to span multiple RTS/control iterations) nor are *synchronized* to external measurement equipment that captures additional (e.g., analog voltage/current) data. Depending on the RTS' communication interfaces and processing pipelines, they might be limited to *internal sources* of temporal events, which leaves relevant I/O latencies (caused by, e.g., GigE controllers) out of reach. Combined latency and power monitoring (to drive both temporal analyses and energy optimizations of RTSs) thus requires not only extended capture lengths including I/O events but also synchronized state and power traces. Due to their *large on-chip memories* and complex capture logic, ELAs might also impact the temporal characteristics of the RTS (e.g., preventing timing closure at a given frequency). Lastly, traditional voltage and current monitoring via external sources or on-board supply controllers is prone to additional *inaccuracies* due to integrated DC/DC converters and restricted acquisition capabilities of the, e.g., ZC702's three UCD9248 (Fig. 2) controllers, respectively. Their limitation in both temporal (with a best-case readout rate of approx. 1 kSPS if only one controller is queried) and current resolutions (with a granularity of approx. 15 mA) stems from the fact that their integrated circuitry is designed for secondary, i.e., slow overcurrent detection [61]. Dedicated power monitors (e.g., TI's INA231 [62]) offer better performance, but still rely on an externally controlled readout over a rate-limited I^{2}C link.

Synchronization of independently acquired (temporal/functional) states or events to analog data is a non-trivial task, given today's system complexity. Whilst online approaches including those implementing long-term, but trace-only acquisition could theoretically be extended to analog data, their required external (off-chip) storage solutions [63] are not only expensive but also proprietary, impeding the integration of custom IP or circuitry. Offline synchronization, on the other hand, can (potentially) be a viable solution – either via established correlation techniques from, e.g., cryptology (such as differential power analysis [64]) or by simultaneously acquiring a shared (i.e., trigger) signal for subsequent shifting in both ELA trace and voltage/current data. Without such a continuous and explicit synchronization source or dedicated fabric logic with a (flexible) measurement system, the independent sampling clocks will cause drifting time bases.

D. Relation to the Proposed Methodology and Quantifications

In comparison, our hybrid solution increases and unifies the joint, overall observability of RTS power/states at significantly *higher levels of abstraction*. This holds true for both resolution and depth of temporal/functional traces required to sufficiently, i.e., system/application envelope-wise, cover the RTS behavior for power/latency analysis and/or optimization. As the *periods, deadlines and execution times* of today's real-time applications range from hundreds of µs to dozens of ms [15], [22], the high, single-cycle (i.e., few ns) resolution from ELA-based solutions is not required. Similarly, single-instruction and per-signal data from CPUs and FPGA fabric is far more accurate than needed. This holds true for the acquisition both of *temporal events* such as completed reception of a sensor value, start of processing (at the real-time pipeline's various stages) or finished transmission of an actuation signal, and of *functional aspects* (e.g., sequence IDs, sensor/actuator data or internal states such as CPU usage). Although sufficient for such application-level monitoring (that, in addition, covers the relatively slow effect of various system-level power management techniques such as DVFS), individual implications of CPU or, e.g., memory operations are discarded. For FPGA-based RTSs, this possibility for abstraction together with the option to instantiate *custom trace IP cores* within their fabric enables the introduction of our hybrid power/state traces that combine energy with macro-phase application monitoring.

In contrast to ELA-based approaches, the proposed trace not only synchronously captures voltages and currents on the RTS' supply rails in an online fashion (Sec. II-C), but also consumes far less FPGA resources due to its level of abstraction and off-chip storage. As our EMS is capable of power monitoring both for multiple rails – including those of I/O components – and at significantly higher temporal/analog resolution than traditional power controllers, the analog data contains timing-relevant I/O events and per-component/phase energy figures all in one. As a similarity to ELAs, the FPGA signals of interest currently have to be selected at design time, although solutions for later trace-insertion could be used to integrate the proposed trace IP core.

Cost-wise, the proposed trace IP core and our EMS compare as follows to the existing solutions. In terms of *FPGA resource usage*, our trace IP core requires less than 500 slice registers (if all implementation options are enabled) as shown in Tab. I. We use a relatively simple configuration (single-bit trigger, 10 data bits) of Xilinx' ILA core as a baseline (for ELA-driven latency monitoring as evaluated in app. B). It requires three times more logic resources and 26x the memory, compared to our solution. VSS pipeline, additional I/O cores and AXI crossbars (Xbars), all part of our reference design, utilize less BlockRAM in total. If trace readback (Sec. IV-D) is disabled, the logic and memory requirements of our core are even lower (Tab. II, no RX FIFO). In *monetary* terms, the hardware is one order of magnitude less expensive than a commercial Data Acquisition (DAQ) solution such as, e.g., a National Instruments PXIe-6124 used as analog quantification reference. Even without its host PC and licenses, a PXI system (app. A) costs over 50x more – albeit commercial manufacturing and support would also increase the EMS price.

TABLE I
COST (I.E., RESOURCE USAGE): VSS PIPELINE VS. ILA VS. TRACE CORE

Component / Subsys.	Slice LUTs	Slice Regs	Slices	BRAM[1]
Base system (Xbars)	5599	7504	2496	0
Additional I/O cores	6256	9375	2757	11
VSS (HW) Pipeline	5456	5679	2140	26
ILA (GMII@PHY2)	1193	1580	619	104
Tracing[2] & IRQ-FF	468	493	188	4
Zynq 7Z020	53200	106400	13330	280

NB: [1] 18 Kbit each / [2] SPIB resource usage differs from Tab. II due to larger RX FIFO

Quantitative analyses of analog/temporal accuracies attained with our approach and external measurement system are shown in appendices A (National Instruments PXI) and B (Xilinx ILA).

III. DESIGN CONCEPTS/PRINCIPLES FOR HYBRID TRACING

On a conceptual level, the proposed methodology relies on five key design principles to facilitate a unified temporal, functional and energy monitoring for current FPGA-accelerated RTSs. As introduced in Sec. II-D, application-level *RTS Instrumentation* for lightweight acquisition of hardware/software pipeline states enables a subsequent reconstruction of temporal and functional system characteristics. Key indicators on the software-side are, e.g., CPU loads, queue fill levels, number of runnable threads, iteration counts, (varying) control gains and even live sensor or actuator signals, although for VSS cases, the former could only be tapped after object detection. Hardware-wise, both multi-bit (e.g., sequence IDs, counters and finite-state machine registers) and single-bit (such as interrupt request) signals are of interest. State probes (dashed red in Fig. 1) forward the relevant signals to our resource-efficient trace IP core, instantiated in the RTS's fabric. In addition to the IP core (orange+blue, bottom center), Fig. 2 also shows the PL (light gray) with its underlying fabric resources. With the CPUs and various other components in the fixed-function PS (dark gray), these DUT resources are used to implement the required heterogeneous real-time pipelines. The trace core converts the state signals into a serial data stream for transmission to an external measurement system (green, right), which requires a suitable interface – and enough analog inputs.

These (in case of our EMS: up to 18) channels are connected to the DUT's on-board supply system (e.g., DC/DC converters) to capture voltages/currents of the required rails. On the ZC702 used in our experiments, for instance, three complex UCD9248 power controllers (Fig. 2, blue) drive ten independent supplies, three of which are selected for *precise RTS Energy Monitoring*, as shown in the center of Fig. 2 (between DUT and UCD9248). The three voltage (dotted) and current (dash-dotted) probes tap the respective rails and thus enable our EMS to monitor supply accurately via the UCD9248s' downstream current sense (CS).

Our EMS also implements *Monitoring of I/O components* by capturing their per-rail power consumption. Based thereon, I/O events such as sensor reception over GigE PHYs are identified. Measuring currents is the most generic implementation of I/O monitoring and applicable to all communication standards that require an external (and thus reachable) PHY or controller, and enables fast turnarounds, e.g., without extra network analyzers.

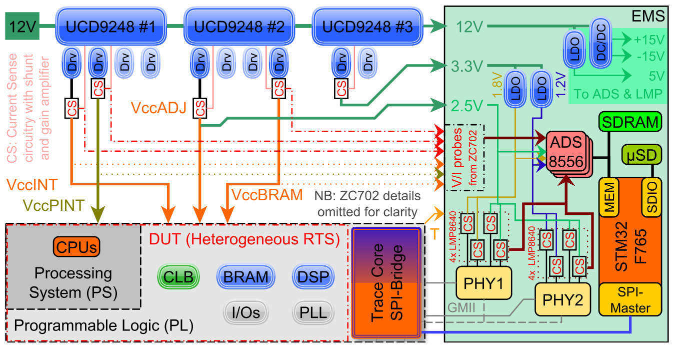


Fig. 2. ZC702 Development Board (left) with a Zynq-based DUT (bottom left) instrumented via our Trace IP Core (center) and External Measurement System (right) using Serial State Stream (blue) and Supply Probes (dotted/dash-dotted)

Precise Timestamping of each acquired analog or state value with sub-µs resolution in our EMS completes the hybrid traces. It enables the subsequent conversion of the power/state data to temporal and I/O events, precise energy figures via integration, and functional information – altogether characterizing an RTS.

Automated RTS Analysis based on the hybrid traces not only enables identification of application phases, estimation of best-case energy savings and a holistic latency monitoring, but also unifies, simplifies and thus significantly shortens RTS analysis.

IV. PROPOSED HARDWARE/IP/SOFTWARE ARCHITECTURE

Our hardware, IP and software architecture implementing the proposed hybrid power/state-tracing features the following key components. The DUT is a heterogeneous RTS based on, e.g., a Xilinx Zynq pSoC, which implements latency- and/or power-constrained, mixed-hardware/software real-time pipelines. Precise per-rail power samples are acquired by our EMS (Sec. III), which taps the on-board shunts of the ZC702 [65] and fuses the state data from our trace core, transferred over a low-pin-count Serial Peripheral Interface (SPI). The resulting hardware architecture including the heterogeneous, Zynq-based RTS (bottom left) on its development board is shown in Fig. 2. In Sec. IV-A, we present the proposed trace IP core and its interfaces to DUT and a generic, SPI-capable measurement system, whilst details of our FMC-sized EMS implementation are given in Sec. V-A. Flexible instrumentation options for tracing of hardware (PL) and software (PS) are shown in Sec. IV-B and IV-C. In case the hybrid power/latency traces should automatically be sent to the RTS after acquisition ended, Sec. IV-D presents a solution with near-zero software overhead including required Linux drivers.

A. SPI-Bridge and DUT/Trace Interface

The primary design goals for the pSoC trace IP core and its surrounding infrastructure were as follows. Both temporal and functional trace information needs to be aggregated and subsequently forwarded with sampling rates and capture precisions (i.e., data widths) sufficient for application-level monitoring of the majority of current RTSs. With such systems' control loops rarely exceeding frequencies above one kHz, a *state sampling rate in the tens of kHz* thus enables capturing dozens of state changes within one control period. The method's high level of abstraction (Sec. II-D/III) also reduces the amount of data that needs to be stored in one sample. With the Visual Servoing Application scenario (Sec. V-C) in mind, a *state size of 16 bytes* is selected, although it can easily be adapted to others. Additional design goals are *low resource utilization* and *minimized DUT interference* – not only to make our methodology viable even close to design completion (where most FPGA resources might be allocated to the actual application), but also to maintain timing closure and I/O assignment in case of tight constraints and heavily utilized pin budgets. Lastly, interfaces to the DUT with its fixed-function PS and configurable PL and the external measurement system should be as generic/flexible as possible.

We thus implement *byte-wide trace ports* (for connection to, e.g., PS GPIOs/IRQs or PL signals) and an optional *AXI slave interface* [66] for fast software instrumentation using memory-mapped I/O (MMIO) writes. As these accesses are un-cached, their impact on timing is minimal and near-constant. To use as few I/O pins as possible whilst relying on only unidirectional signals (due to speed and integrity considerations), we propose a *three-wire SPI*-like connection between our trace IP core and external measurement system. In case the AXI slave is enabled during IP configuration, the *split between PS and PL trace* data can be set on a byte-wise basis. The generic trace-ports also include a per-byte acknowledge signal, which can be used to reset simple *upstream preprocessors* in the PL – such as edge-capture blocks or application-specific (e.g., activity) counters.

Fig. 3 shows the basic structure of the trace IP core with its interfaces to both DUT and external measurement system. Its gateway to the latter is a simple 8-bit shift register connected to the SPI – that consists of two inputs, Serial Clock (SCLK) and Master Out Slave In (MOSI), and a single output, Master In Slave Out (MISO). The trace information captured from the DUT is buffered in a 16-byte transmit (TX) fabric memory and shifted out synchronously to the outside SCLK. Alternatingly, incoming data streams (sent by an EMS) can optionally be forwarded to the DUT by including additional receive (RX) logic for trace readback (Sec. IV-D). As the core effectively bridges DUT traces over SPI, it was quickly coined SPI-Bridge (SPIB).

This efficient structure enables our EMS with its hardware-triggered SPI master functionality to retrieve DUT state traces, which it then merges with its own analog (i.e., voltage/current) measurements to form the hybrid power/state trace. The latter is thus stored externally, reducing DUT resource consumption and efforts for subsequent analysis (as the fused trace is readily available without further, offline post-processing). As commercially available DAQ solutions (in contrast to our FMC-sized implementation) commonly rely on software-based SPI, their precise clock sources and cross-input synchronization features cannot directly be used for synchronization. In this case, state timestamps have to be generated via complex post-processing.

In retrospect, the resulting infrastructure performs as follows w.r.t. the above design goals. With the central SPI shift register supporting external (SCLK) frequencies beyond 50 MHz even on slowest speed-grade devices and the chosen sample size of 16 bytes, the theoretical state capture rate readily approaches 400 kSPS – which is even sufficient for most high-rate control systems. If the desired RTS monitoring does not demand such

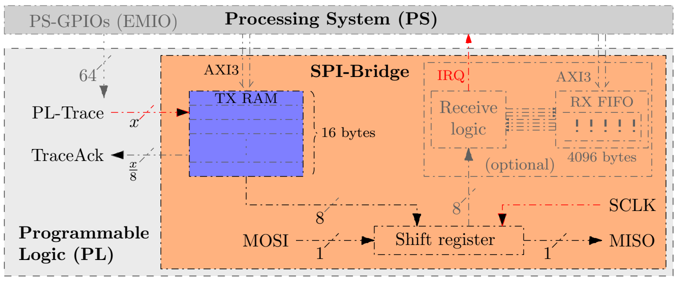


Fig. 3. Internals and Interfaces of the Trace IP Core (SPI-Bridge, center right)

dense state samples, the available bandwidth can either be utilized for larger state samples, left unused or be reduced by means of the SPI clock frequency. Both resource consumption and the resulting interference with the DUT hardware (PL) are extremely small, as even the fully-fledged implementation including trace readback to the PS requires less than 500 slice registers of a Xilinx Zynq 7Z020, which is below 5‰ of the device resources [67]. The routing overhead to tap PL signals of interest, however, depends on the particular architecture and might necessitate intermediate capture registers in tight designs like with ELAs and other tracing methodologies. The SPI-based link between trace core and external measurement system only uses three I/Os, whilst the internal ports enable various implementation options for interfacing to PS and PL.

B. Implementation Options for PS & PL

The proposed hardware/software architecture consisting of DUT, trace IP core and an external measurement system can easily be tailored to the tracing/debugging scenario at hand. This not only holds true for the byte-wide trace ports tapping the hardware and software components (implemented in PL and PS) of the RTS, but also for an independent AXI interface for trace readback from the DUT. Whilst this section focuses on configuration options of the trace IP core and acquisition of PL states, Sec. IV-C presents options for PS tracing – whereas the software infrastructure in kernel and userspace for trace readback from our measurement system follows in Sec. IV-D.

In its most basic form, the SPIB implements only a single, 16-byte shift register with associated control logic to reload or update the entire register (after 128 bits have been clocked out by the external master). Although extremely resource-efficient, this option – at first glance – only allows tracing of states from the PL. On Zynq pSoCs, however, the signals of the general-purpose I/O (GPIO) controllers (part of the PS) can be routed to the PL by means of the Extended Multiplexed I/O (EMIO) interface that connects PS and PL [68]. As indicated in Fig. 3, up to 64 bits of PS trace data could thus be written to the SPIB without additional logic resources. Analogously, this holds true for Intel's devices, which support routing of peripheral signals from HPS to FPGA fabric [69]. In case all these fixed-function GPIOs are unavailable, an *AXI slave interface* can be enabled within the SPIB to make a configurable number of byte-wide trace ports available from software. The segmentation of the (at the moment 16) data bytes into PS and PL is given in Fig. 4. As more than eight bytes of state data were never required, we use

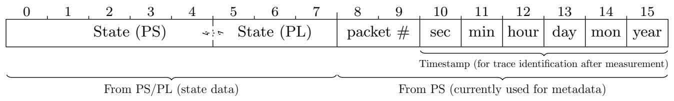


Fig. 4. Segmentation of Trace Data within the SPI-Bridge (currently 16 Bytes)

the remaining half to transfer selected metadata to our EMS at system initialization. This, e.g., enables time synchronization from DUT software to EMS, which simplifies the identification of stored traces based on their date- and timestamp during later analysis. In both cases, the software running on the PS (i.e., either bare metal applications or, in case an operating system is used, kernel and userspace code) can easily be instrumented to update the trace data via MMIO-based write accesses. Multiple methods for tracing from Linux userspace (that do not require development of custom device drivers) are shown in Sec. IV-C.

Trace Readback: In addition to transmitting state samples, the SPIB can be configured to handle incoming data transfers (i.e., from an external measurement system to DUT). We use this feature to download the hybrid power/state trace recorded by our FMC-sized measurement system to the DUT after the acquisition is complete. Once received, the trace is available as a regular file in Linux. This not only simplifies iterative, manual measurements as the trace thus can easily be transferred using, e.g., `scp`, but also enables fully automated, DUT-driven measurement cycles where each trace is stored alongside other performance-related data of the individual iteration. Details of the required hardware and software are presented in Sec. IV-D.

Preprocessing: If the PL signals of interest change at similar or even higher frequencies than the state capture rate, the resulting trace will be incomplete due to aliasing. Depending on the particular tracing/debugging scenario, however, the data rate of certain types of signals can often be reduced without loss of information relevant for application-level monitoring of the RTS. We explored implementation options for both rare, single-shot signals such as interrupt request (IRQ) lines and also the per-cycle activity or enable signals of many IP cores as found in, e.g., image processing (Sec. V-C). For the former, we implemented a byte-wide *edge-detection preprocessor* that resides between the various IRQ signals, which are required to inform the PS CPUs that an IP core in the PL needs their attention. Effectively equal to an 8-bit set/reset flip-flop, it only requires five slices within the PL and runs at full clock rate of the design. In case an IRQ line is asserted even for a single cycle, the corresponding bit remains set until this event has been seen by the SPIB, which afterwards resets the preprocessor by means of the TraceAck signal (Fig. 3). The addition of IRQ events to the trace data provides valuable information regarding the temporal behavior of the heterogeneous RTS, in particular for Direct Memory Access (DMA) IRQs that often indicate a migration of processing from PS to PL or vice versa. Activity/enable signals of, e.g., image processing IP cores, on the other hand, can be used to trace utilization during regular system operation. This, among others, enables fast estimation of best-case power savings for activity-dependent management techniques such as clock gating (Sec. VI-C). Thus, the average duty cycle of such signals is sufficient, which similarly can be captured using a small *activity counter* also automatically reset after the number of cycles with asserted enable signal has been captured by the SPIB and acknowledged via its TraceAck port.

TABLE II
RESOURCE USAGE OF THE SPI-BRIDGE (FULLY FEATURED)

Component	Slice LUTs	Slice Regs	Slices	BRAM[1]
Control	55	55	30	0
TX RAM	8	0	2	0
RX FIFO[2]	69	100	38	2
Interconnect[3]	328	330	126	0
SPI-Bridge	460	485	186	2
Zynq 7Z020	53200	106400	13330	280

NB: [1]in 18 Kbit mode (RAMB18E1) / [2]trace readback / [3]MMIO/readback

Tab. II shows the resource usage of the SPI-Bridge and its individual components with all features enabled compared to the total device resources of a medium-sized Zynq 7Z020. The resource usage is dominated by an AXI interconnect sourced from Xilinx [70], which is required to convert the PS' complex AXI3 interface to the reduced AXI4-Lite inside the SPIB and thus has to be factored into the overall resource consumption.

C. Kernel/Userspace Tracing via MMIO

If either a (PS-internal) GPIO or the SPIB's AXI slave interface is used to transfer state information (from PS to EMS), the software (executed by the PS CPUs) is easily instrumented due to the MMIO-based interface of both options. In case the application runs bare metal, i.e., without an operating system (OS), plain write accesses to GPIOs or SPIB are sufficient. If an OS' kernel implements an *unprivileged userspace*, kernel code still may use direct hardware access itself. Normal applications on, e.g., Linux, however, have to rely on kernel interfaces such as GPIO/LED subsystems, userspace I/O or `/dev/mem` [71]–[74].

D. Low-Overhead Trace Readback to PS

Again, we implement this feature with resource efficiency in mind, but also take software aspects into account. The receive channel consists of a small control logic and a BlockRAM for data buffering. As long as acquisition is running, the external measurement system continuously transmits zeros (on MOSI) whilst retrieving the trace data (via MISO). Afterwards, the state trace (starting with a non-zero header) is automatically uploaded. The non-zero value on MOSI triggers the receive logic to assert its IRQ line and store the incoming data stream in the BlockRAM until reset from the PS. On the software (i.e., PS) side, we assign the `generic-uio` driver to the SPIB (by devicetree modification [75]). This driver is part of the stock Linux kernel release and, due to our design of the receive logic, sufficient to both inform a small userspace daemon of the incoming data and expose the receive buffer to userspace via MMIO. After initialization, the daemon blocks while waiting for the IRQ and thus does not consume any CPU time in this state. As measurement and transfer are mutually exclusive, the (predominantly) dormant daemon has negligible impact on the PS (i.e., software execution) and is only scheduled on demand.

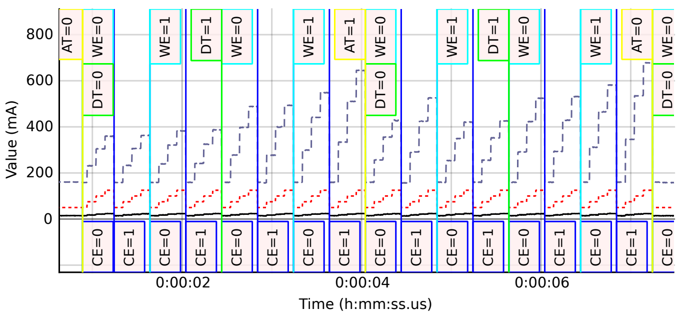


Fig. 5. Hybrid Power/State Trace of Synthetic PL Workload: Fabric & BRAM Currents (dashed/gray & solid/black), Test Size (dotted red) & Control Signals

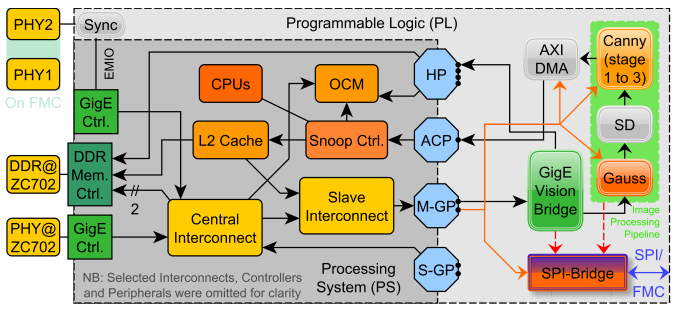


Fig. 6. Visual Servoing Scenario on Zynq: PS I/Os (left), PS (center) and PL with hardware-based Image Acquisition/Processing & State-Tracing via SPIB

V. Reference EMS Implementation and Use Cases

We use our cost-effective FMC-sized measurement system (see Sec. V-A for a feature summary or [40, 41] for implementation details), specifically designed to capture multi-rail analog (i.e., voltage/current) values and state samples synchronously – thus generating the proposed hybrid traces, which unify all required measurements into a single workflow with very short (<1 min.) and thus debugging-friendly turnaround times. To demonstrate their benefits, we evaluate a synthetic PL workload (Sec. V-B) and a real-world VSS (Sec. V-C), using a ZC702 and our EMS.

A. *FMC-based Measurement Subsystem*

One of the ZC702 ports connects to our FMC-sized external measurement system that features not only an 18-channel, 16-bit-resolution acquisition subsystem capable of over 200 kSPS, but also two independent GigE PHYs [76], [77] – whose power consumption is also monitored to identify sensor and actuator I/O. As Ethernet is increasingly used for RTS interfacing, this enables end-to-end (i.e., input-to-output) latency monitoring. Internally, the acquisition subsystem routes three synchronized analog-to-digital converters (ADCs) [78], gain amplifiers [79] and additional circuitry to a Cortex M7 microcontroller (Fig. 2, right) [80]. The latter relies on SDRAM [81] and µSD flash to store analog samples and serial state stream received using one of its integrated SPI master peripherals. Due to performance limitations of external data bus, controller and µSD, the overall logging throughput may not exceed ≈14 MByte/s. This results in a trade-off between sampling rates of analog and state data, which is currently set to a ratio of 8:1 analog to state samples.

During acquisition, the firmware (running on the Cortex M7 microcontroller, standalone without OS) continuously polls all three ADCs as fast as possible, merges the state data received via an SPI master (at above ratio) and adds sub-µs timestamps to each ADC/SPI sample to create the hybrid power/state trace.

B. *Synthetic Fabric & BRAM Workload*

This first, synthetic application scenario aims to identify the power demand (across two PL rails) of workloads that heavily utilize the FPGA fabric's memory structures (i.e., BlockRAM). We thus rely on a simple RTL address/data generator to induce intense switching activity on the inputs/outputs of 240 BRAMs (out of a total 280). During the test, the BRAMs are enabled in three groups of 80 as indicated by the dotted red line in Fig. 5. In addition, we not only toggle Write (`WE`) and Output Register Enables (`CE`), but also control the switching activity on both address (`AT`) and write data (`DT`) busses. The BRAMs are configured for 9-bit address and full 36-bit data widths using write-first, registered output mode and are clocked at 50 MHz.

As expected, this causes drastic current variations on the PL rail (dashed gray) driving both traffic generation logic and, as apparent from the barely loaded BRAM rail (solid black), also a significant portion of the BRAMs. Together, analog and state data allow an automated baseline characterization (Sec. VI-B).

C. *Visual Servoing Application Scenario*

As mixed-hardware/software real-time pipelines are particularly hard to analyze w.r.t. their temporal behavior (Sec. II-C), we also apply the proposed methodology to a VSS application scenario – featuring a hybrid PS/PL/PS processing pipeline [2]. A GigE Vision camera [82] continuously captures the height of a magnetic levitating sphere, serving as representative use case for high-speed VSSs. The control goal is to maintain its stable levitation at a configurable position by altering the current sent through an overhead magnetic coil. At 178 frames per second, the incoming camera data cannot be handled in software alone. Our design thus not only maps a part of the image processing pipeline (Gaussian filter [83], Canny edge detection [84], contour finding [85] and object detection) to the PL, but also relies on purely hardware-based image acquisition for the incoming GigE Vision video stream. This acquisition is implemented via a custom IP core that interfaces the fixed-function GigE controller within the PS to *receive all* and – subsequently – *extract the video*-related Ethernet frames. Whilst proxying non-image frames back into the PS, this GigE Vision Bridge (GEVB) core then feeds the pixel stream (byte-wise) to the Image Processing Pipeline (IPP) also implemented in the PL. Fig. 6 shows both selected PS internals – including the GigE controller used for image acquisition via PHY2 – and the VSS pipeline consisting of GEVB, IPP and a Xilinx DMA core for PS writeback once an entire camera image has traversed the PL hardware pipeline.

The remaining processing steps (i.e., starting from hysteresis edge tracking for Canny) are then performed by our userspace application running on the PS CPUs, which also computes and transmits the actuation signal. Our custom Linux kernel driver manages the transfer of both control and video data from a PL DMA controller to userspace via the Video4Linux subsystem.

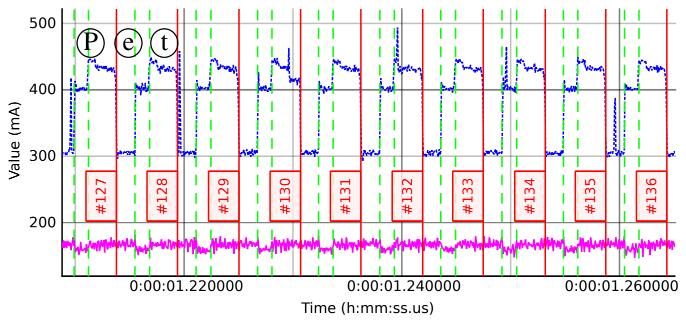


Fig. 7. Hybrid Power/State Trace of the Visual Servoing System: PS & Fabric Currents (dotted/blue & solid/pink), Image-IDs (#red) & -Input (dashed green)

Fig. 7 shows the processing of ten frames and, unlike Fig. 8, also includes the required fabric current (solid pink). It slightly increases during reception of camera images, i.e., between the first payload frame (Ⓟ) and the handover from PL to PS (ⓔ), whereas end of CPU execution (ⓣ) yields a falling PS current.

VI. Applications and Benefits of Hybrid Tracing

As our hybrid traces combine multi-rail power with accurately synchronized PS/PL states, they cover a variety of properties in a single, unified measurement and are widely applicable during development/optimization of today's FPGA-accelerated RTSs.

For our experiments with PL workload and Visual Servoing System, we monitored the following supply rails on the ZC702 board and both GigE PHYs. The Zynq's VCCPINT (PS cores), VCCINT (PL fabric) and VCCBRAM (PL memory) rails are acquired independently (i.e., both voltage and current), leaving 12 channels for I/O monitoring. We capture VCCADJ (=2.5 V) and VCC3V3 (both from ZC702) – plus our FMC board's 1V2 and 1V8, which feed the Ethernet PHYs' core and analog rails. With four supplies (two analog, one each for I/O and core) per PHY, the remaining eight channels are entirely used for shunts.

A. Identification of Application Phases

The majority of current real-time and everyday applications comprises multiple sequential phases of processing. This holds particularly true for pipelined, mixed-software/hardware scenarios such as games or web browsers, which offload rendering to a GPU – and most RTSs due to their inherent input-process-output structure. The identification of all individual application phases is crucial to answer functional (i.e., *What did it do and in which order?*) and temporal (*When did it do it?*) questions during debugging and/or optimization. Balancing latencies and power of a mixed-hardware/software application, for instance, requires precise knowledge of both – for each processing phase and every relevant (hardware/software) component of the RTS.

Whilst even offline (e.g., current-driven) phase identification is relatively straightforward in case of the synthetic workload due to its predefined, self-contained nature, the real-world VSS poses a significant challenge. This is because it not only relies on hardware and software components with unknown temporal and energy characteristics (such as all PS interconnects and the Linux kernel), but also interacts with the RTS's outside world (i.e., camera and actuator) over an – event-triggered – Ethernet.

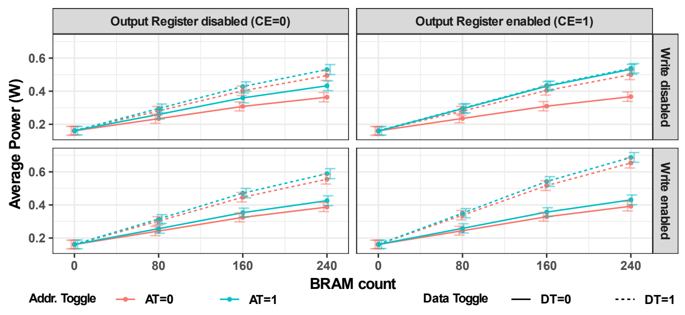


Fig. 9. Synthetic PL Workload: Fabric Power versus Test Size in 2^4=16 Cases

Although the application's periodic nature is clearly indicated by the supply rails of PS (dotted blue) and fabric (solid pink) in Fig. 7, both temporal and functional details such as acquisition latencies, processing times and pipeline behavior (e.g., overlap) are out of reach. Including all details captured by our hybrid trace, however, the internals of even a single application iteration quickly become clear (Fig. 8). Implementing the GigE Vision Stream Protocol (GVSP), the camera first transmits its GVSP leader frame – which is then received by the RTS, as indicated by a small spike on PHY2's I/O rail (gray, magnified by 10x). Once processed by the PS GigE controller, it traverses the PS and reaches the PL for detection by the GEVB (Ⓛ). The following group of 56 GVSP payload frames (carrying the actual image data) is automatically identified based on PHY2's rail and extends from Ⓟ to ⓔ (dashed green lines and `PHY:` markers). As GEVB and IPP operate in-stream (i.e., without buffering entire images), the hardware pipeline finishes shortly after the last payload frame (ⓔ) and processing migrates back to the PS – i.e., its CPUs, which execute Linux and userspace application (Ⓢ). The latter not only asserts the (one-bit wide) trigger line (cyan) notifying the external measurement system, but also inserts the ID of the just received video image into the state trace using software instrumentation (red `CPU: Start` marker). As soon as both the remaining image processing steps and control computation are completed, the actuation data is finally transmitted (ⓣ). Together, the long camera readout (Ⓛ to ⓔ) and CPU-based image/control computations (Ⓢ to ⓣ) extend the overall latency beyond the control period of approx. 5.6 ms – Iteration #103 thus starts before iteration #102 ended.

It should be noted that – due to the entirely hardware-based image acquisition – all hardware-related timestamps (Ⓛ, Ⓟ and ⓔ) are out of reach of traditional, software-based tracing. As the former records the point in time of the camera image's arrival, it is essential to deduce the total (i.e., end-to-end) RTS latency (Sec. VI-D) required to ensure control stability (Sec. I). Thanks to the temporal and functional coverage of our hybrid trace, the control designer immediately becomes aware that the pipeline overlap prohibits a traditional, "zero delay" approach. Instead, advanced – delay-aware – strategies are required, even though the VSS pipeline's relatively small jitter (cf. Sec. VI-D) might just obviate control designs for time-varying delays [86].

In case the PHYs' power consumption is not of interest, the I/O monitoring can be reduced to two channels (Sec. VI) – i.e., the I/O rail currents instead of all supply voltages and currents.

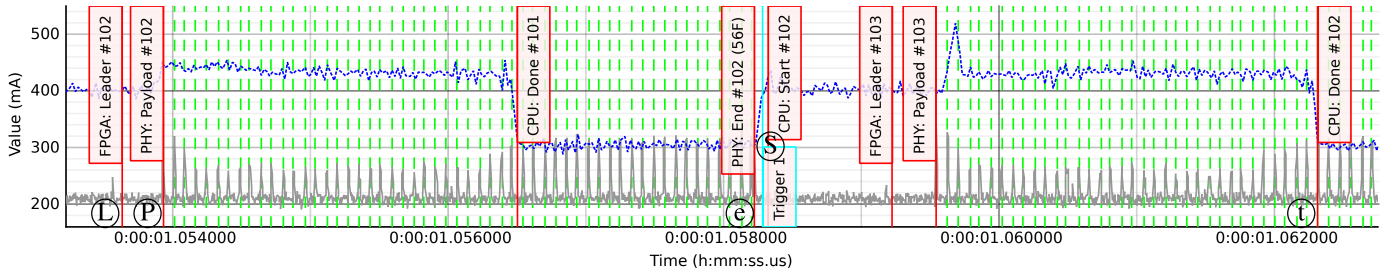


Fig. 8. Hybrid Power/State Trace with PS & PHY Currents (dotted/blue & solid/gray/10x), every incoming Ethernet Frame (dashed green) & Image-IDs (red)

B. *Phase/Component Energy Baselines*

Each of the *synthetic workload*'s (Sec. V-B) 2^4=16 test cases incrementally activates the BRAMs' enable signals in groups of 80. Even in the first case (`AT=WE=DT=CE=0` – solid orange line in upper left plot of Fig. 9) without switching activities, a significant increase in power consumption can be seen, which corresponds to the first three steps in Fig. 5 (left). In addition to only instantiating as few BRAMs as possible (to benefit from the automatic power gating on modern FPGAs, cf. Sec. II), it thus is also advisable to (dynamically) disable each individual BRAM whilst not accessed. As expected, increased switching activities (i.e., assertion of `AT`/`WE`/`DT`/`CE`) also lead to a higher power consumption – the only exception being the transition of `WE=0` to `1` in test case `AT=1`/`DT=0`/`CE=1` (i.e., solid cyan line on the right of Fig. 9). Moving from top (`WE=0`) to bottom (`WE=1`), the power consumption – to the authors' surprise – in fact decreases. Although the reasons are currently unclear, it also should be noted that running at a higher clock rate would result in even steeper curves due to a linear relation to power.

In the *Visual Servoing Application Scenario* (Sec. V-C), we again focus on the power consumption of the PL fabric. Based on the activity timestamps gathered from the state trace (i.e., Ⓟ and ⓔ), we identify an average PL power consumption of 159 mW in idle and 167 mW during frame reception (Fig. 7/8). Also shown in Tab. III, we captured another hybrid power/state trace whilst sequentially disabling all the clocks driving the PL (`FCLKx`, generated within the PS). This yields a residual static power consumption of 40 mW – equivalent to savings of 76%. It, however, should be noted that this is an absolute best-case value assuming that the entire PL fabric could be clock-gated, effectively rendering the VSS inoperative as incoming Ethernet frames can no longer be received by the PL (for filtering by the GEVB), thus initially queue and eventually drop within the PS.

C. *Estimated Best-Case Energy Savings*

For a more realistic estimation of potential best-case savings of the VSS-related PL components, both their fabric utilization and activity ratio are required. Whilst the former can easily be calculated as 29.3% based on utilization reports in Vivado [87], the latter depends on various run-time aspects such as spacing of individual GVSP payload frames, behavior of the PS GigE controller and writeback rates of the DMA controller in the PL.

We thus use activity signals of GEVB and downstream IPP cores to enable a small PL-based *activity counter* running at full design frequency of 125 MHz – as proposed in Sec. IV-B. As its present value can easily be acquired using the SPIB (red dashed wires in Fig. 6), the resulting state trace thus includes piece-wise, cycle-accurate utilization data of the VSS pipeline.

Based on the measured activity ratio of 15.6%, the pipeline thus could be gated 84.4% of the time. The estimated best-case power savings thus are 76% · 29.3% · 84.4% = 18.8% of the active power consumption – yielding savings of 31.4 mW. Such accurate estimations of potential savings require not only synchronized state information (i.e., IPP activity), but also per-phase energy values – both beyond traditional tools (Sec. II-C).

TABLE III
LATENCY AND POWER READINGS DERIVED FROM THE HYBRID TRACE

Period and/or Phase	In Fig. 7/8	Latency	Jitter/σ	PL Power
Inter-Image @ PHY	Ⓛ → Ⓛ	5.62 ms	18.87 µs	–
Inter-Image @ CPU	Ⓢ → Ⓢ	5.62 ms	19.25 µs	–
Reception of Image	Ⓟ → ⓔ	4.29 ms	10.42 µs	167 mW
PL: Waiting for RX	ⓔ → Ⓟ	–	–	159 mW
PL-IPP & IRQ→PS	ⓔ → Ⓢ	0.12 ms	17.46 µs	–
PS: IPP, Ctrl. & TX	Ⓢ → ⓣ	3.81 ms	54.30 µs	–
RTS total (In→Out)	Ⓛ → ⓣ	8.53 ms	55.02 µs	–
PL Idle (`FCLKx`=0)	(Sec. VI-B)	–	–	40 mW

NB: Selected latency/power readings were omitted for clarity (but are part of the trace)

D. *Phase, Component and I/O Latencies*

Based on all the events gathered from the hybrid power/state trace (Sec. VI-A), crucial temporal characteristics of the RTS can be quantified – as summarized in Tab. III that also includes the measured jitter given in terms of the standard deviation (σ). This not only includes the pure-software execution time (Ⓢ to ⓣ) with 3.81 ms, but also mixed-hardware/software figures such as the DMA interrupt response time (ⓔ to Ⓢ) of 120 µs. Unsurprisingly, the overall end-to-end jitter ("RTS" in Tab. III) is largely caused by software ("PS") – particularly by the edge-dependent execution time of the IPP's contour finding step. In combination with PHY-based events, image transmission time (Ⓟ to ⓔ) of 4.29 ms (σ=10 µs) and – most importantly – the total RTS latency, i.e., from image reception to transmission of actuation signal (Ⓛ to ⓣ), with 8.53 ms (σ=55 µs) are found.

This value is not only crucial for control engineering (as it represents the system's dead time that needs to be compensated during the design of the control algorithm), but also enables us to evaluate the performance of various proprietary (i.e., black-box) IPs such as GigE controllers part of the datapath (Fig. 6).

VII. Conclusion

In this paper, we proposed hybrid power/state-tracing as a unified, yet generic methodology to not only capture temporal and functional characteristics of complex FPGA-accelerated RTSs, but also correlate these properties to the energy consumption of various system components. We presented a resource-efficient trace IP core with a cost-effective measurement system to both gather state streams from the RTS, and fuse them to the multi-rail power measurements. Such hybrid traces enable automated identification of various temporal events within complex – i.e., mixed-hardware/software – real-time application pipelines that are crucial to resolving functional, temporal and energy-related questions. In combination with the analog measurements, these events permit the computation of per-phase and per-component energy baselines – which are essential for subsequent, accurate power/latency optimization and best-case energy estimation. In addition, they also yield extensive latency coverage required to ensure stability of control loops closed by the RTS, as demonstrated for a heterogeneous, real-word Visual Servoing System. Beyond the end-to-end latencies and potential best-case energy savings, the trace data also unveiled its severe pipeline overlap.

Jointly, the proposed trace IP core and external measurement system implement a complete solution for (holistic) RTS monitoring. It – in particular due to its simplicity – is effective (w.r.t. costs/resources) and practical – and already has proven its wide applicability to further application and measurement scenarios.

For future work, we intend to use the proposed methodology not only to perform efficiency comparisons between bare metal and Linux implementations of the VSS, but also for combined latency- and power-optimization of PS (via frequency scaling) and PL (using fine-grained clock gating) processing stages. We expect traditional (i.e., software-driven) power management to be insufficient due to the high control rate – thus necessitating a mixed-software/hardware management module whose development and evaluation will greatly benefit from hybrid tracing.

Appendix

We performed exhaustive measurements to ensure the accuracy of our EMS (Sec. V-A) generating the hybrid power/state trace.

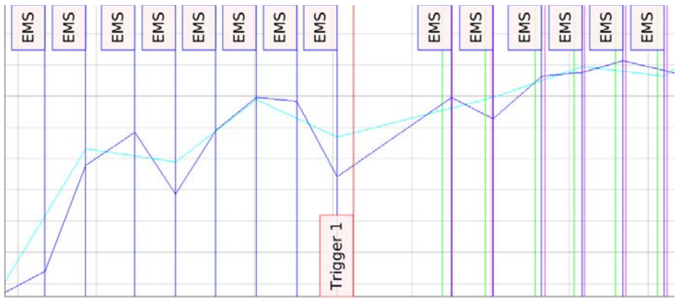


Fig. 10. Timestamping Accuracy: EMS vs. PXI (blue vs. pink) and PS Current

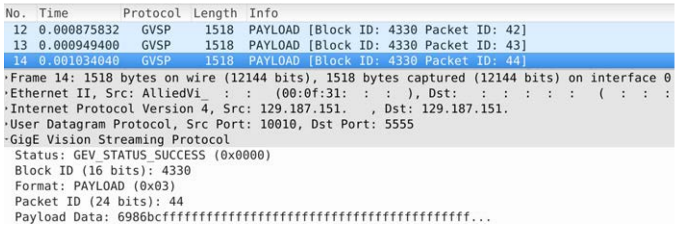
A. *Quantitative Comparison to Energy Monitoring via NI-PXI*

We use the National Instruments PXIe-6124 4-channel DAQ card [88] for a quantitative evaluation of our solution's analog and temporal accuracies. In addition to its four, simultaneously sampling 16-bit ADCs, it features 24 GPIO channels – that can also be acquired or used as clock/trigger sources, within limits. The (synchronized) analog and digital inputs sample at 4 MHz, and, once relayed from PXI chassis over PCIe, reach a host PC.

Fig. 10 shows the rising PS current (Ⓢ in Fig. 8) at a zoom level high enough to recognize individual EMS samples (blue vertical markers). To quantify temporal inaccuracies (i.e., the interval between the recorded sampling times and actual ADC acquisitions), we synchronized EMS and PXI by means of the DUT trigger line. In addition, we logged the ADC busy signals using PXI GPIOs, and plotted rising (green) and falling (pink) edges. It can be easily seen that the EMS shortly pauses analog sampling due to the incoming trigger (red) and the firmware's best-effort behavior (Sec. V-A). Subsequently, the timestamps of analog acquisitions recorded by EMS (blue) and PXI (pink) show only minor offsets – confirmed by a mean error of 0.2 µs.

Although state samples (received via the SPIB) benefit from the same timestamp resolution, their sampling rate of currently 25 kSPS imposes a maximum uncertainty of 40 µs on the state data – i.e., stored transitions might have occurred ≤40 µs ago.

To quantify analog (i.e., voltage/current – and thus indirectly power) measurement errors, we use two analog channels of the PXI (in differential setups) to accurately capture the PS supply rail. Whilst voltage is directly measured (probing `C138` on the ZC702), current has to be captured indirectly (by means of the shunt voltage across `R76`). Our measurement systems taps into the ZC702's own instrumentation amplifier (an INA333, `U45`), whereas the PXI is driven from a dedicated, internally trimmed INA213, piggybacked onto the ZC702 for maximum accuracy. Our solution records a PS energy consumption of 21.46 mJ for 10 VSS iterations. With a PXI reading of 21.53 mJ, a relative error of -0.3% is found (including all temporal/analog factors).



```
No. Time         Protocol Length Info
 12 0.000875832   GVSP     1518  PAYLOAD [Block ID: 4330 Packet ID: 42]
 13 0.000949400   GVSP     1518  PAYLOAD [Block ID: 4330 Packet ID: 43]
 14 0.001034040   GVSP     1518  PAYLOAD [Block ID: 4330 Packet ID: 44]
·Frame 14: 1518 bytes on wire (12144 bits), 1518 bytes captured (12144 bits) on interface 0
·Ethernet II, Src: AlliedVi_  :  :   (00:0f:31:  :  :  ), Dst:   :  :  :  :  :   (  :  :  :
·Internet Protocol Version 4, Src: 129.187.151.   , Dst: 129.187.151.
·User Datagram Protocol, Src Port: 10010, Dst Port: 5555
·GigE Vision Streaming Protocol
  Status: GEV_STATUS_SUCCESS (0x0000)
  Block ID (16 bits): 4330
  Format: PAYLOAD (0x03)
  Packet ID (24 bits): 44
  Payload Data: 6986bcffffffffffffffffffffffffffffffffffffffffff...
```

Fig. 11. Timestamping Accuracy: 14 Ethernet Frames in Wireshark (via ELA)

B. *Quantitative Comparison to Latency Monitoring with ELAs*

As evaluation of our solution's temporal accuracy for events, e.g., I/O activity, and as comparison to the functional/temporal monitoring capabilities of ELAs, we instantiate Xilinx' System ILA [47] in the PL. Driven by the receive clock, it is able to tap the Gigabit Media-Independent Interface (GMII) of PHY2 (see Fig. 2/6) and record all raw incoming Ethernet frames on a per-cycle basis (i.e., at 125 MHz). Using ≈37% of BRAM (Tab. I), the ILA is able to capture 14 Ethernet frames containing image data. After preprocessing, we import all frames into Wireshark to find the transfer latency of 1.03404 ms in Fig. 11. Measuring the same interval in our hybrid trace (Fig. 8) yields 1.03333 ms and thus an extremely low relative error of 0.7‰ from our tool.

References

[1] 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)*.

[2] 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)*.

[3] R. Wilhelm, J. Engblom, A. Ermedahl, N. Holsti, S. Thesing, D. Whalley, G. Bernat, C. Ferdinand, R. Heckmann, T. Mitra, F. Mueller, I. Puaut, P. Puschner, J. Staschulat, and P. Stenström, “The worst-case execution-time problem – overview of methods and survey of tools”, *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 7, no. 3, pp. 36:1–36:53, May 2008.

[4] M. Geier, T. Burghart, M. Hackl, and S. Chakraborty, “In situ latency monitoring for heterogeneous real-time systems”, in *2019 32nd International Conference on VLSI Design and 2019 18th International Conference on Embedded Systems (VLSID)*.

[5] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, “Low-power cmos digital design”, *IEEE Journal of Solid-State Circuits*, vol. 27, no. 4, pp. 473–484, Apr. 1992.

[6] 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.

[7] M. Bao, A. Andrei, P. Eles, and Z. Peng, “On-line thermal aware dynamic voltage scaling for energy optimization with frequency/temperature dependency consideration”, in *2009 46th ACM/IEEE Design Automation Conference (DAC)*.

[8] 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)*.

[9] S. DSouza, A. Bhat, and R. Rajkumar, “Sleep scheduling for energy-savings in multi-core processors”, in *2016 28th Euromicro Conference on Real-Time Systems (ECRTS)*.

[10] 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.

[11] 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].

[12] 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].

[13] S. Huda, M. Mallick, and J. H. Anderson, “Clock gating architectures for fpga power reduction”, in *2009 19th International Conference on Field Programmable Logic and Applications (FPL)*.

[14] 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)*.

[15] 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.

[16] D. Ramanathan, S. Irani, and R. Gupta, “Latency effects of system level power management algorithms”, in *2000 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*.

[17] S. Irani, S. Shukla, and R. Gupta, “Online strategies for dynamic power management in systems with multiple power-saving states”, *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 2, no. 3, pp. 325–346, Aug. 2003.

[18] E. Grochowski, R. Ronen, J. Shen, and H. Wang, “Best of both latency and throughput”, in *2004 22nd IEEE International Conference on Computer Design: VLSI in Computers and Processors (ICCD)*.

[19] E. Nilsson and J. Öberg, “Reducing power and latency in 2-d mesh nocs using globally pseudochronous locally synchronous clocking”, in *2004 2nd IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*.

[20] P. T. Congdon, P. Mohapatra, M. Farrens, and V. Akella, “Simultaneously reducing latency and power consumption in openflow switches”, *IEEE/ACM Transactions on Networking (TON)*, vol. 22, no. 3, pp. 1007–1020, Jun. 2014.

[21] P. Radojković, S. Girbal, A. Grasset, E. Quiñones, S. Yehia, and F. J. Cazorla, “On the evaluation of the impact of shared resources in multithreaded cots processors in time-critical environments”, *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 8, no. 4, pp. 34:1–34:25, Jan. 2012.

[22] F. Wartel, L. Kosmidis, C. Lo, B. Triquet, E. Quiñones, J. Abella, A. Gogonel, A. Baldovin, E. Mezzetti, L. Cucu, T. Vardanega, and F. J. Cazorla, “Measurement-based probabilistic timing analysis: Lessons from an integrated-modular avionics case study”, in *2013 8th IEEE International Symposium on Industrial Embedded Systems (SIES)*.

[23] C. Emde, “Long-term monitoring of apparent latency in preempt_rt linux real-time systems”, in *2010 12th Real-Time Linux Workshop (RTLWS)*.

[24] T. Amert, N. Otterness, M. Yang, J. H. Anderson, and F. D. Smith, “Gpu scheduling on the nvidia tx2: Hidden details revealed”, in *2017 38th IEEE Real-Time Systems Symposium (RTSS)*.

[25] A. Horga, S. Chattopadhyay, P. Eles, and Z. Peng, “Measurement based execution time analysis of gpgpu programs via se+ga”, in *2018 21st Euromicro Conference on Digital System Design (DSD)*.

[26] S. Alcaide, L. Kosmidis, C. Hernandez, and J. Abella, “High-integrity gpu designs for critical real-time automotive systems”, in *2019 22nd Design, Automation and Test in Europe Conference (DATE)*.

[27] K. Irrgang and T. B. Preußer, “An lz77-style bit-level compression for trace data compaction”, in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*.

[28] N. Decker, B. Dreyer, P. Gottschling, C. Hochberger, A. Lange, M. Leucker, T. Scheffel, S. Wegener, and A. Weiss, “Online analysis of debug trace data for embedded systems”, in *2018 21st Design, Automation and Test in Europe Conference (DATE)*.

[29] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, “Towards energy proportionality for large-scale latency-critical workloads”, in *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*.

[30] H. Kasture, D. B. Bartolini, N. Beckmann, and D. Sanchez, “Rubik: Fast analytical power management for latency-critical systems”, in *2015 48th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[31] C.-H. Chou, D. Wong, and L. N. Bhuyan, “Dynsleep: Fine-grained power management for a latency-critical data center application”, in *2016 International Symposium on Low Power Electronics and Design (ISLPED)*.

[32] V. Devadas and H. Aydin, “On the interplay of voltage/frequency scaling and device power management for frame-based real-time embedded applications”, *IEEE Transactions on Computers (TC)*, vol. 61, no. 1, pp. 31–44, Jan. 2012.

[33] K. Huang, L. Santinelli, J. Chen, L. Thiele, and G. C. Buttazzo, “Adaptive dynamic power management for hard real-time systems”, in *2009 30th IEEE Real-Time Systems Symposium (RTSS)*.

[34] Xilinx Inc., “Xilinx Power Estimator User Guide”, https://www.xilinx.com/cgi-bin/docs/rdoc?v=2019_2;d=ug440-xilinx-power-estimator.pdf, 2019, [UG440; Revision v2019.2 of October 30, 2019].

[35] 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.

[36] 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)*.

[37] S. M. Afifi, F. Verdier, and C. Belleudy, “Power estimation method based on real measurements for processor-based designs on fpga”, in *2014 International Conference on Computational Science and Computational Intelligence (CSCI)*.

[38] Xilinx Inc., “7 Series FPGAs and Zynq-7000 SoC XADC Dual 12-Bit 1 MSPS Analog-to-Digital Converter”, https://www.xilinx.com/support/documentation/user_guides/ug480_7Series_XADC.pdf, 2018, [UG480; Revision 1.10.1 of July 23, 2018].

[39] C. T. Chow, L. S. M. Tsui, P. H. W. Leong, W. Luk, and S. J. E. Wilton, “Dynamic voltage scaling for commercial fpgas”, in *2005 IEEE International Conference on Field-Programmable Technology (FPT)*.

[40] M. Geier, D. Faller, M. Brändle, and S. Chakraborty, “Cost-effective energy monitoring of a zynq-based real-time system including dual gigabit ethernet”, in *2019 IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[41] M. Geier, D. Faller, M. Brändle, and S. Chakraborty, "Cost-effective energy monitoring of a zynq-based real-time system including dual gigabit ethernet", Mar. 2019, arXiv:1903.09548 [eess.SP].

[42] G. Bernat, R. Davis, N. Merriam, J. Tuffen, A. Gardner, M. Bennett, and D. Armstrong, "Identifying opportunities for worst-case execution time reduction in an avionics system", *Ada User Journal*, vol. 28, no. 3, pp. 189–194, Sep. 2007.

[43] The Real-Time Linux Collaborative Project, "realtime:start [Wiki]", https://wiki.linuxfoundation.org/realtime/start, 2020.

[44] Intel Corporation, "Intel 64 and IA-32 Architectures Software Developer Manuals", https://software.intel.com/en-us/articles/intel-sdm#uncore, 2019, [Uncore Performance Monitoring Reference Manuals; Revision of November 11, 2019].

[45] A. Yasin, "A top-down method for performance analysis and counters architecture", in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[46] OSADL - Open Source Automation Development Lab eG, "Latency Box", https://www.osadl.org/Latency-Box.projects-latency-box.0.html, 2016.

[47] Xilinx Inc., "System Integrated Logic Analyzer v1.0", https://www.xilinx.com/support/documentation/ip_documentation/system_ila/v1_0/pg261-system-ila.pdf, 2017, [PG261; Revision 1.0 of June 7, 2017].

[48] Intel Corporation, "Quartus Prime Pro Edition User Guide – Debug Tools", https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug-qpp-debug.pdf, 2019, [UG-20139; Revision 2019.09.30].

[49] Z. Poulos, Y. Yang, J. Anderson, A. Veneris, and B. Le, "Leveraging reconfigurability to raise productivity in fpga functional debug", in *2012 15th Design, Automation and Test in Europe Conference (DATE)*.

[50] E. Hung and S. J. E. Wilton, "Incremental trace-buffer insertion for fpga debug", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, vol. 22, no. 4, pp. 850–863, Apr. 2014.

[51] E. Hung and S. J. E. Wilton, "Speculative debug insertion for fpgas", in *2011 21st International Conference on Field Programmable Logic and Applications (FPL)*.

[52] E. Hung and S. J. E. Wilton, "Limitations of incremental signal-tracing for fpga debug", in *2012 22nd International Conference on Field Programmable Logic and Applications (FPL)*.

[53] J. S. Monson and B. Hutchings, "New approaches for in-system debug of behaviorally-synthesized fpga circuits", in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*.

[54] J. Goeders and S. J. E. Wilton, "Using dynamic signal-tracing to debug compiler-optimized hls circuits on fpgas", in *2015 IEEE 23rd Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[55] J. Goeders and S. J. E. Wilton, "Signal-tracing techniques for in-system fpga debugging of high-level synthesis circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 36, no. 1, pp. 83–96, Jan. 2017.

[56] Y. Choi and J. Cong, "Hlscope: High-level performance debugging for fpga designs", in *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*.

[57] J. Goeders and S. J. E. Wilton, "Using round-robin tracepoints to debug multithreaded hls circuits on fpgas", in *2015 International Conference on Field Programmable Technology (FPT)*.

[58] A. Kourfali and D. Stroobandt, "Efficient hardware debugging using parameterized fpga reconfiguration", in *2016 30th IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*.

[59] I. Ahmed, A. Kamaleldin, H. Mostafa, and A. N. Mohieldin, "Utilizing dynamic partial reconfiguration to reduce the cost of fpga debugging", in *2018 16th IEEE International New Circuits and Systems Conference (NEWCAS)*.

[60] F. Eslami and S. J. E. Wilton, "An adaptive virtual overlay for fast trigger insertion for fpga debug", in *2015 International Conference on Field Programmable Technology (FPT)*.

[61] Texas Instruments Incorporated, "Digital PWM System Controller", https://www.ti.com/lit/gpn/UCD9248, 2012, [SLVSA33; Revision A].

[62] Texas Instruments Incorporated, "INA231 High- or Low-Side Measurement, Bidirectional Current and Power Monitor With 1.8-V I²C Interface", https://www.ti.com/lit/gpn/ina231, 2018, [SBOS644; Revision C].

[63] Agilent Technologies Inc., "On-Chip Design Verification with Xilinx FPGAs", https://literature.cdn.keysight.com/litweb/pdf/5988-9434EN.pdf, 2003, [AN1456 (5988-9434EN); Revision of April 30, 2003].

[64] Q. Tian and S. A. Huss, "A general approach to power trace alignment for the assessment of side-channel resistance of hardened cryptosystems", in *2012 8th International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*.

[65] Xilinx Inc., "Zynq-7000 SoC - ZC702 Evaluation Kit", https://www.xilinx.com/support/documentation-navigation/design-hubs/dh0039-zc702-evaluation-kit-hub.html, 2019.

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

[67] Xilinx Inc., "Zynq-7000 SoC Data Sheet: Overview", https://www.xilinx.com/support/documentation/data_sheets/ds190-Zynq-7000-Overview.pdf, 2018, [DS190; Revision 1.11.1 of July 2, 2018].

[68] 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 1.12.1 of July 1, 2018].

[69] Intel Corporation, "Stratix 10 Hard Processor System Technical Reference Manual", https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/s10_5v4.pdf, 2020, [s10_5v4; Revision 2020.01.25].

[70] Xilinx Inc., "AXI Interconnect", https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf, 2017, [PG059; Revision 2.1 of December 20, 2017].

[71] The kernel development community, "GPIO Sysfs Interface for Userspace", https://www.kernel.org/doc/html/v5.5/admin-guide/gpio/sysfs.html, 2019.

[72] The kernel development community, "LED handling under Linux", https://www.kernel.org/doc/html/v5.5/leds/leds-class.html, 2019.

[73] H.-J. Koch, "The Userspace I/O HOWTO", https://www.kernel.org/doc/html/v5.5/driver-api/uio-howto.html, 2019.

[74] The Linux man-pages project, "mem(4) - Linux manual page", https://www.kernel.org/doc/man-pages/online/pages/man4/mem.4.html, 2015.

[75] Linaro Limited, "Specifications - DeviceTree", https://www.devicetree.org/specifications/, 2020, [Current Revision: v0.3 of 13 February 2020].

[76] Texas Instruments Incorporated, "DP83865 Gig PHYTER V 10/100/1000 Ethernet Physical Layer", https://www.ti.com/lit/gpn/DP83865, 2004, [SNLS165; Revision B].

[77] Microchip Technology Inc., "KSZ9031MNX - Gigabit Ethernet Transceiver with GMII/MII Support", https://ww1.microchip.com/downloads/en/DeviceDoc/00002096E.pdf, 2017, [DS00002096; Revision E].

[78] Texas Instruments Incorporated, "ADS855x 16-, 14-, 12-Bit, Six-Channel, Simultaneous Sampling Analog-to-Digital Converters", https://www.ti.com/lit/gpn/ADS8556, 2016, [SBAS404; Revision D].

[79] Texas Instruments Incorporated, "LMP8640/-Q1/HV Precision High Voltage Current Sense Amplifiers", https://www.ti.com/lit/gpn/LMP8640, 2014, [SNOSB28; Revision G].

[80] STMicroelectronics, "STM32F765xx (...) MCU+FPU", https://www.st.com/resource/en/datasheet/stm32f765zi.pdf, 2017, [029041; Revision 6].

[81] Integrated Silicon Solution, Inc., "IS42/45S81600F, IS42/45S16800F: 16Mx8, 8Mx16 128Mb Synchronous DRAM", http://www.issi.com/WW/pdf/42-45S81600F-16800F.pdf, 2019, [Revision D1].

[82] Allied Vision Technologies GmbH, "Mako G-032 Gigabit Ethernet camera with Sony ICX424 CCD sensor", https://www.alliedvision.com/en/products/cameras/detail/g-032.html, 2016.

[83] R. Cope and P. Rockett, "Efficacy of gaussian smoothing in canny edge detector", *Electronics Letters*, vol. 36, no. 19, pp. 1615–1617, Sep. 2000.

[84] J. Canny, "A computational approach to edge detection", *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 8, no. 6, pp. 679–698, Nov. 1986.

[85] S. Suzuki and K. Abe, "Topological structural analysis of digitized binary images by border following", *Computer Vision, Graphics, and Image Processing (CVGIP)*, vol. 30, no. 1, pp. 32–46, Apr. 1985.

[86] 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.

[87] Xilinx Inc., "Vivado Design Suite User Guide: Design Analysis and Closure Techniques", https://www.xilinx.com/cgi-bin/docs/rdoc?v=2019_2;d=ug906-vivado-design-analysis.pdf, 2019, [UG906; Revision v2019.2 of October 30, 2019].

[88] National Instruments Corp., "NI 6124 Specifications", https://www.ni.com/pdf/manuals/372526b.pdf, 2008, [372526B; Revision 01 of Dec08].