VIADUCT: Bridging the Gap between Testbeds and Real-World Cyber-Physical Systems*

Philipp Sommer
ABB Research
Baden-Daettwil, Switzerland
philipp.sommer@ch.abb.com

Felix Sutton
ABB Research
Baden-Daettwil, Switzerland
felix.sutton@ch.abb.com

Abstract

The success of the fourth industrial revolution hinges on the ability to deploy robust cyber-physical systems that monitor and control physical processes. Testbeds are state of the art tools for evaluating such systems according to the execution requirements of the application, *i.e.*, evaluating how the cyber-physical system interacts with the device upon which it is implemented on. However, the validation of a cyber-physical system must also consider the reactive requirements of the application, which define how the cyber-physical system interacts with the underlying physical process it is designed to monitor and control. We advocate that a new testbed infrastructure is needed that facilitates the evaluation of execution and reaction requirements simultaneously.

We present the design and implementation of VIADUCT, a novel testbed infrastructure that models the physical process under consideration, and injects stimulus into the cyber-physical system using time synchronized hardware interfaces. An experimental evaluation demonstrates how the testbed simultaneously evaluates execution and reaction requirements with a precision and scalability that surpasses the capabilities of state of the art testbeds, thereby bringing testing infrastructure one step closer to evaluating real-world cyber-physical systems.

Categories and Subject Descriptors

Computer systems organization [**Embedded and cyber-physical systems**]: Sensor networks

General Terms

Design, Experimentation, Measurement, Performance

Keywords

Testbed, virtual sensor, virtual actuator, power profiler

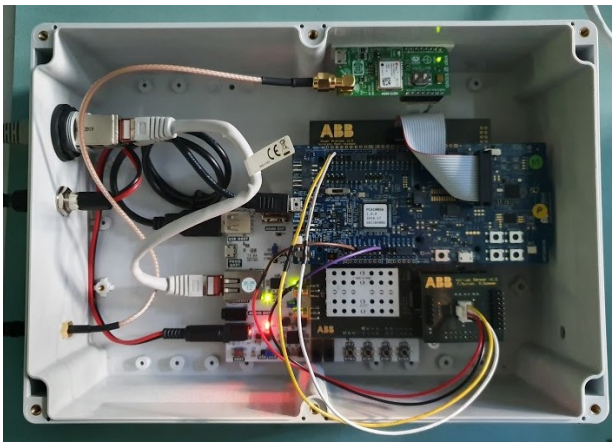


Figure 1: A VIADUCT observer with a Nordic nRF52840-DK target. The observer provides *virtual sensors and actuators*, which model the underlying physical process that the cyber-physical system is designed to monitor and control.

1 Introduction

The success of Industry 4.0 hinges on the ability to seamlessly integrate devices that are capable of interacting with real-world processes. These systems, commonly referred to as cyber-physical systems (CPS), combine computation, communication, sensing and actuation to monitor and control an underlying physical process, such as the vibration of an industrial motor, the temperature of a high-power electrical distribution bus, or the flow rate of a catalyst in a chemical process. In order to verify the behavior of these complex systems, rigorous testing infrastructures and methodologies are needed.

Problem. Testbeds, as surveyed in [24], are the state of the art tool for testing cyber-physical systems. These testbeds have facilitated significant research and development of cyber-physical systems through the availability of public testbeds, such as D-Cube [28], Indriya2 [10], and Flocklab [22], and dependability competitions [7, 27]. However, the problem is that these testbeds evaluate the device under test in complete isolation, *i.e.*, the complex interaction between the device and the physical process is not considered. The ramifications of this may be severe, as when the

*Both authors contributed equally to this research.

International Conference on Embedded Wireless Systems and Networks (EWSN) 2021
17–19 February, Delft, The Netherlands
ISBN: 978-0-9949886-5-2

cyber-physical system is deployed into the real-world, the intricate interactions with the underlying physical process being monitored or controlled may adversely affect the performance of the system, *e.g.*, through increased power dissipation, increased latencies or erroneous hardware and/or software states. We therefore advocate the need to extend the scope of testing to also include the complex interaction between the device under test and the physical world.

Challenges. Clearly, it is infeasible to bring a physical process, such as the vibration monitoring of industrial machines, into a laboratory testbed. And similarly, it is impractical to bring an embedded systems laboratory into a harsh industrial environment. Instead, we need to model the important characteristics of the physical process under consideration, and embed the model into the testbed infrastructure.

The first challenge is to define a suitable level of abstraction, *i.e.*, at what level of abstraction shall we model the physical process such that it closely resembles the behavior of a real-world deployment. Once we have a model, we need to create a testbed architecture that can implement this model with a high degree of measurement precision, thus enabling the accurate calculation of performance indicators such as average power dissipation, end-to-end latency and software state mapping. Finally, the testbed architecture must scale to multiple devices, as cyber-physical systems typically contain networks of devices, while also supporting the testing over long time scales so to effectively evaluate long-term performance characteristics.

These three challenges, *i.e.*, modeling, precision and scalability, are interdependent. For example, a model may incorporate a slow moving process, such as daily temperature variation, and in order to evaluate the performance of the device under test, precision measurements must be performed over long time scales. As precision measurements are achieved through high-rate sampling, *e.g.*, of voltage and current, a large amount of data produced significantly limits the scalability of the testbed infrastructure. We therefore need to address these challenges by modeling the physical process using an appropriate level of abstraction, coupled with an architecture that supports a high degree of precision, without sacrificing scalability.

Approach. We propose VIADUCT, a new testbed architecture that bridges the gap between modern testing infrastructure and a real-world deployment of a cyber-physical system. VIADUCT models the underlying physical process using a hardware-level abstraction layer, termed *virtual sensors* and *virtual actuators*, based on commonly-used digital interfaces such as Inter-Integrated Circuit (I2C), Inter-IC Sound (I2S) and Serial Peripheral Interface (SPI). This unique abstraction layer makes it possible to inject complex stimuli throughout the cyber-physical system representative of a real-world deployment. Using well-established system design principles [19, 20, 21], we partition VIADUCT into components with well-defined interfaces, and map them onto appropriate analog circuits, digital blocks, soft-cores and dedicated processors. The unique architecture of VIADUCT ensures that precision measurements are achieved while supporting long duration testing of several days for cyber-physical systems containing scores of devices.

Contributions. The contributions of this paper are as follows:

- We introduce a new abstraction of physical processes, termed *virtual sensors* and *virtual actuators*, which make it possible to bring the complex characteristics of a real-world cyber-physical system into a testbed infrastructure.
- We present the architecture of VIADUCT, a new testbed that closes the gap between modern testbeds and a real-world deployment of a cyber-physical system.
- We detail the design and implementation of a VIADUCT testbed with 16 nodes, one of which is depicted in Fig. 1, and experimentally evaluate its unique features using a set of representative micro-benchmarks.
- We present a case study that shows how VIADUCT can be used as a tool to facilitate testing of a cyber-physical application by providing virtual acoustic signals to the device under test.

This paper is structured as follows: In Sec. 2, we present the background and related work of modern testbeds. We then introduce the virtual sensor and actuator concept in Sec. 3, detail the architecture of the VIADUCT testbed in Sec. 4 and present its design and implementation in Sec. 5. We experimentally evaluate the features of VIADUCT through a series of micro-benchmarks in Sec. 6 and show VIADUCT in action for a smart building use case in Section Sec. 7, before discussing limitations and future improvements in Sec. 8 and concluding remarks in Sec. 9.

2 Background and Related Work

It is more than 15 years since the first testbed, Motelab [32], was presented in the literature. As illustrated in Fig. 2a, a testbed consists of one or more target nodes, each consisting of a microcontroller, a radio module and a collection of sensors and actuators. Attached to each target is an observer node, which is responsible for *(i)* programming the target, *(ii)* logging the output from the target, *e.g.*, serial logs, GPIO traces, voltage and current measurements, and *(iii)* triggering input into the target, *e.g.*, the rising and falling of selected GPIO lines.

It is important to highlight that the device under test, *i.e.*, the set of target nodes, are isolated from the underlying physical process that the targets are designed to monitor and control. Despite this fundamental limitation, the testbed concept has been widely used in academia. In particular, testbeds have been extensively used in the research of wireless sensor networks, where they have been instrumental in the performance evaluation of wireless protocols. An extensive range of testbeds have been proposed in the literature to date, including low-cost testbeds D-Cube [28] and OpenTestBed [25], large-scale testbeds TWIST [17], Kansei [12], Indria [10], and Indriya2 [2], the FlockLab [22] testbed having multi-target support, a mobility-enabled testbed FIT IoT-LAB [1], the Minerva [31] testbed providing advanced online debugging tools, the TempLab [9] for emulating environmental temperature variations, JamLab [8] and JamLab-NG [28] for injecting interference into a wireless network, the Shepherd [16] testbed for transiently-powered targets,

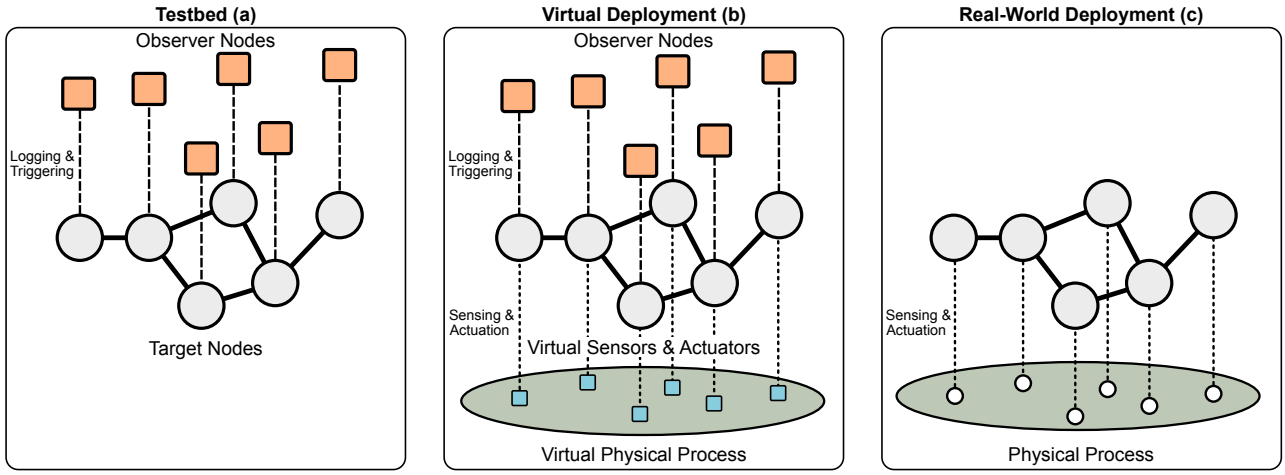


Figure 2: A graphical representation of a cyber-physical system in a testbed (a), a VIADUCT virtual deployment (b), and a real-world deployment (c).

and recently, the LinkLab testbed [15] for remote development and testing of IoT applications. We acknowledge the importance of these testbeds and support the ongoing initiatives to create a set of benchmarks for the evaluation of wireless sensor networks [11, 5, 4].

Testbeds are primarily designed to evaluate the *execution requirements* of a cyber-physical system. These execution requirements measure the performance of the target upon which the CPS is built upon. Examples include the power dissipation of the microcontroller, the end-to-end latency of the wireless protocol, and the tracing of software state. However, cyber-physical systems not only have execution requirements, but they also have *reactive requirements* [18], which are equally important. These reactive requirements measure the performance of the cyber-physical system in response to interactions with the underlying physical process.

As illustrated in Fig. 2c, once a cyber-physical system is deployed into the real-world, each target uses its sensors and actuators to monitor and control the underlying physical process. These interactions with the physical process are complex. The cyber-physical system must react to changes in the physical process, which may originate from changes monitored at a single target, or may be attributed to actions performed by one or more targets within the network. For example, the adjustment of a solenoid valve at one target in a chemical process application may influence the flow rate measured by other targets. This may result in a burst of high-priority network traffic, leading to collisions and the invocation of retransmission mechanisms, thus leading to increased latency and higher power dissipation that may destabilize the entire cyber-physical system.

This motivates the need to model the underlying physical process in a local and distributed manner, as illustrated in Fig. 2b. VIADUCT adopts the features of modern testbeds while also modeling the physical process found in a real-world deployment. VIADUCT supports the logging and triggering akin to modern testbeds with high precision and scalability, and through appropriate abstractions, models the underlying physical process such that the reactive requirements

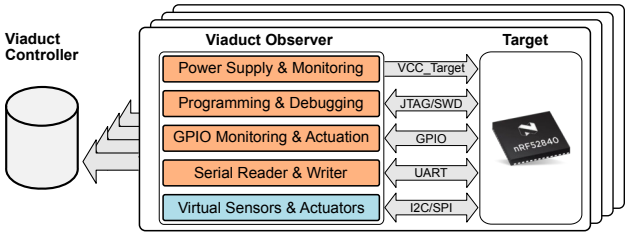


Figure 3: Blueprint of VIADUCT's system architecture: Each observer is connected to a target device for programming & debugging, GPIO monitoring & actuation, serial logging and power profiling. Furthermore, we provide virtual sensors and actuators connected to the target. A network of several observers is connected to the central testbed controller.

of the cyber-physical system can be fully understood and evaluated prior to a real-world deployment.

An overview of the VIADUCT architecture is shown in Fig. 3. Each target device, which we have chosen to be the Nordic nRF52840 system-on-chip, is connected to a VIADUCT observer. The VIADUCT observer provides a range of services, which include supplying the target with power and measuring its power consumption, monitoring and actuating GPIO lines, reading from and writing to the serial interface, and virtualizing sensors and actuators. Multiple VIADUCT observers may be deployed in a wired and/or wireless IP network, thus enabling each observer to connect to a VIADUCT controller. The controller is responsible for scheduling testbed jobs, as well as storing and visualizing the output of each job.

3 Virtual Sensors and Actuators

In this section, we describe how VIADUCT's capabilities can be utilized for testing embedded software applications that interact with the physical world using virtual sensors and actuators.

Cyber-Physical Systems. The key aspect in which CPS differ from traditional computing systems is the fact that computational tasks are linked with the physical world through

sensors and actuators. Feedback loops exist between computation and physical systems where the outcome of a software task affects the behavior of the real-world environment and vice versa. For example, the embedded system will initiate a communication round with other nodes in the network triggered by an external event observed through its sensors.

Challenges. Due to the tight coupling between computational and physical parts of the system, testing the embedded system alone without the physical process in the loop will not be sufficient to emulate and predict the behavior during the real-word deployment phase. We argue that exploring the complex interactions between the physical and computational components already during the design and implementation phases by the means of virtual deployments with real-hardware is an integral part of the overall design process.

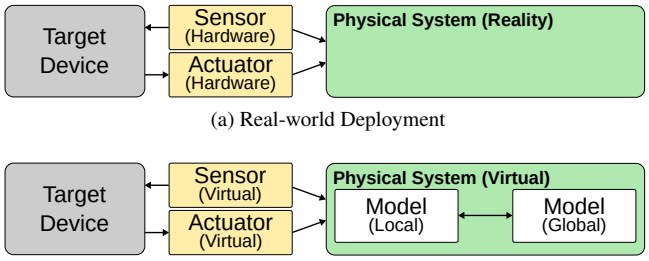
Emulating the Physical World. VIADUCT allows to emulate the physical part of a complex CPS system in order to explore the behavior of the embedded system part interacting with it. In order to provide a flexible way to emulate the physical world part of a CPS, we introduce the concept of *virtual* sensors and actuators. While a hardware-based sensor converts an observation of the physical world into a voltage signal or digital representation, a virtual sensor provides measurement values based on a software-based model of the environment. Following the same principle, a *virtual* actuator updates the state of a software model of the environment instead of interacting directly with the physical world.

Virtual Sensors and Actuators. Although our sensors and actuators have no electrical components to measure or interact with the physical process, we keep the identical electrical interfaces of such hardware components, as for example the popular UART, I2C or SPI interfaces. Furthermore, we can also employ a digital-to-analog converter (DAC) or an analog-to-digital converter (ADC) to provide analog inputs to the target node, as well as to measure analog voltages produced by the target. Consequently, we can model the interface of virtual sensors and actuators to imitate the behavior of hardware components that will be used in real-word CPS devices. This has the benefit that the timing behavior of the embedded system will not differ between virtual and real hardware, which allows to employ the same binary image during testing with VIADUCT and for the real-world deployment.

CPS Models. As detailed in [26], we assume cyber-physical systems are modeled by multi-dynamical systems, which have the following properties:

- Discrete events based on the outcome of computation
- Continuous physical processes
- Uncertainty due to:
  - Stochastic processes (*e.g.*, noise)
  - Non-deterministic processes
  - Adversarial behavior (*e.g.*, actuation of one influences observation of another node)

The VIADUCT testbed architecture provides the capabilities to model the physical process of CPS with the above mentioned properties. Based on the interactions of the target device with the virtual actuators, we can update our continuous model of the physical environment accordingly, and furthermore, reflect the new model state in the measurements provided by the virtual sensors. Processes with non-deterministic behavior can also be modeled by emulating discrete events using interrupts.



(a) Real-world Deployment

(b) Virtual Deployment

Figure 4: Target devices interact with their environment using sensor and actuator hardware components (a). VIADUCT provides virtual sensors and actuators to emulate the behavior of the physical world using models (b).

Local vs. Global Models. Compared to a real-world deployment, as illustrated in Fig. 4a, we distinguish between models for *local processes*, which only take into account the interactions of a single node with its environment, and models for *global processes*, which model the state of the environment using a global process, as illustrated in Fig. 4b. The system architecture of VIADUCT is agnostic of the model of the environment and supports both local and global models. VIADUCT's message broker allows for flexible and scalable exchange of model state and discrete events between a central instance and the observers as well as between distributed models.

4 Architecture

In this section, we discuss the key requirements for a testbed architecture providing the capabilities to facilitate testing of cyber-physical systems, as discussed in the previous sections.

4.1 Requirements

We derive the following key requirements for the VIADUCT testbed architecture:

Inspection. The testbed should offer several monitoring and debugging options such as serial logging, GPIO tracing and power profiling.

Synchronized Clocks. Collecting measurement data from several spatially distributed observers and several logging modalities (GPIO, serial, power) requires that timestamps associated with the data are aligned to the same time base both on observer as well as network level. Furthermore, coordination of actuation between devices will also require synchronized clocks.

Scalability. Increasing the number of devices under test should not adversely impact the underlying system architecture. Furthermore, the architecture should support to perform tests with a long duration, *e.g.*, multiple days or weeks.

Live Operation and Processing. Visibility into the state of the devices under test should be available as early as possible, *i.e.*, during the test already instead of only after the test.

Extensibility and Flexibility. The architecture should allow to extend the system with additional functionality while not requiring to redesign the overall architecture. Furthermore, the observer should be agnostic of the target hardware as much as possible to support experiments with novel platforms with minimal changes.

Modularity. In order to allow to extend the testbed with additional features, the core functionality of the testbed should be implemented in separate hardware and software modules.

4.2 Design Principles

Designing the VIADUCT testbed architecture is a challenging endeavor, so we therefore aim to manage the complexity using a set of well-established design principles [19, 20, 21]. We next briefly described the principles, and how they are applied to the architecture of VIADUCT.

Abstraction. In order to model the behavior of the physical process in VIADUCT, we must choose an appropriate abstraction layer. This was motivated by identifying that the time-domain is a critical property that needs to be preserved by the abstraction, since physical processes underpinning cyber-physical systems are tightly coupled to time. Furthermore, since sensors and actuators used in real-world deployments are typically interfaced by a digital bus, such as the popular I2C and SPI interfaces, it is natural to define the physical process abstraction layer at the I2C/SPI digital interface, while preserving the bus clock and the response time of the sensors and actuators being modeled.

Partitioning. Also referred to as the separation of concerns or spatial isolation, we seek to partition tasks based on their information flow, *i.e.*, control or data path, and the real-time deadlines, *i.e.*, soft or hard deadlines. Given the nature of the analog-domain, precision voltage and current measurements are performed using custom analog circuits. As for the digital-domain, data-driven tasks with hard real-time deadlines are performed in custom hardware blocks, while control-driven tasks with soft real-time deadlines are performed in soft-cores or dedicated processors.

Segmentation. We next explored if there was a need to employ temporal isolation within the system, that is, are there segments of the system that can be performed sequentially. This decision was primarily motivated by the need for high precision measurements, since the VIADUCT observer must time stamp events, *i.e.*, measurements, serial logs, changes in GPIO line states, etc., with a high sampling rate, these components operate on a localized high frequency time-base, *i.e.*, a 100 MHz free-running clock. This time base is only maintained within the VIADUCT observer. All other components that interact with network services towards the VIADUCT controller are based on the network time-base, *i.e.*, UTC time. In practice, this means the dedicated data-driven hardware blocks operate on the local clock domain, while all other software threads executing on soft- and hard-cores operate on the network clock domain.

Interface Semantics. It is important that components being mapped onto heterogeneous resources, such as software threads executing on a dedicated processor or a soft-core, dedicated hardware blocks or analog circuits, must share information using well-defined interfaces. In particular, dedicated dual-port block random access memory, *i.e.*, shared memory, is used to interface hardware blocks to soft- and hard-cores, a high-speed bus is used to connect dedicated hardware blocks, interrupts are used to distribute control between hardware and software components, while a scalable message broker is used to interface software components.

Component Reuse. We embrace stable open-source hardware and software components and tools. Specifically, we use a Linux distribution with Python scripts coupled with an open source message broker to interface software tasks between observer and controller.

5 Design and Implementation

In this section, we outline how several components, such as observers, message broker, and testbed controller, are composed into the overall system architecture of VIADUCT.

5.1 VIADUCT Observer

The system architecture of the observer is based around the Xilinx Zynq-7020 system-on-chip platform, which integrates two ARM Cortex-A9 processors with a programmable logic (FPGA) part. We employ the Digilent Arty Z7 single-board computer, which hosts the Zynq-7020 and peripherals such as a Ethernet and USB port. The Zynq platform allows us to segment our subsystems into time-critical low-level blocks implemented in the FPGA fabric, while high-level software tasks can be executed in user-space on the Linux OS. Data transfer between the programmable logic and the ARM processors can be achieved using DMA transfers into the DDR RAM or read-/write operations on the internal high-speed memory buses. The chosen hardware platform and software architecture allows us to combine both high-level scripting languages, such as Python, with time-critical hardware blocks implemented in the logic fabric. We run with the unmodified Xilinx Linux kernel tree and employ the Linux user space I/O system (UIO) to provide memory-mapped access to our custom logic cores in the FPGA. Furthermore, user space modules are notified on hardware events using the Linux interrupt system to avoid polling by software.

Target under Test. The target under test is powered by an adjustable voltage supply between 1.8 V and 3.5 V. Dedicated level shifters are employed to ensure the monitoring of GPIO lines, actuation of GPIO lines and logging of UART is supported irrespective of the supply voltage selected. The target may be programmed using standard programming and debug tools, such as SEGGER J-Link and DAPLINK tools, serial bootloader or advanced debug probes. These advantageous features ensure the selection of the target is based on the needs of the cyber-physical system, and is not determined by the constraints of the testbed infrastructure.

Power Profiling. The power profiling of the target, *i.e.*, the measurement of the target supply voltage and current drain over time, is performed in low-side configuration using a precision analog front-end interfaced to an analog to digital converter. The current and voltage sense circuits were adapted from the RocketLogger [30] open source project. The RocketLogger supports two current channels and four voltage channels, however, as the VIADUCT observer supports a single target, only one current and voltage channel are needed. The current channel consists of two current moni-

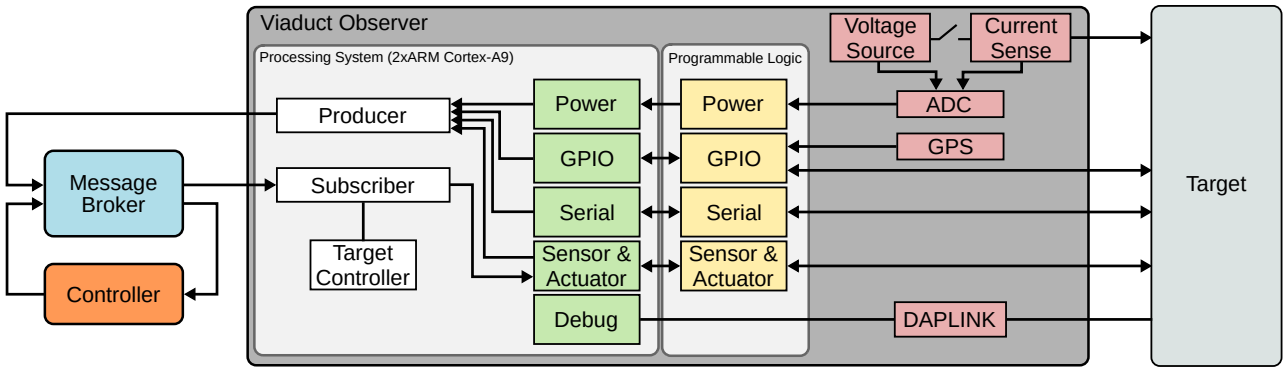


Figure 5: Hardware and software building blocks of the VIADUCT observer: Time-critical operations are handled by dedicated modules implemented in the programmable logic (FPGA), while control and communication is handled by a companion user space software module. The observer is connected to the message broker as a producer for monitoring of GPIO, serial and power. Furthermore, it is a subscriber for messages from the testbed controller to start/stop testbed jobs.

tors in series, one designed for a low range, *i.e.*, between 0 mA and 2 mA, and the other for a high range, *i.e.*, between 2 mA and 500 mA. The low range is based on a feedback ammeter circuit, while the high range is based on a shunt ammeter circuit. The voltage channel consists of a buffered non-inverting amplifier circuit with fixed gain. The current and voltage channels are sampled by an ADS131E08 24-bit analog to digital converter with a default sampling rate of 1 kSPS, but could be configured up to 64 kSPS. The detailed performance metrics of the voltage and current measurement configuration may be found in [29].

UART and GPIO. We support state-of-the art debugging capabilities required for testing by logging the target's serial output and tracing of GPIO pins. The digital output and input line of the target's UART transceiver are connected to a UART transceiver core in the Zynq's programmable logic. The serial logging component implemented in Python reads the target's serial output and forwards it to the message broker together with the corresponding timestamp.

We have implemented a custom GPIO logging and time capture core in VHDL, which supports monitoring transitions in the logical level of up to 16 digital input and output lines. Each rising or falling edge event of a monitored GPIO line is timestamped with a free-running 100 MHz counter and written into a FIFO memory buffer. A companion Python module continually reads out reported events from the buffer and forwards them to the message broker. In order to limit the network bandwidth, we decode the data using the Google Protocol Buffer binary format and combine multiple events into a single message to the broker.

Time Synchronization. High-resolution timestamping of low-level internal and external events is performed using a free-running 32-bit counter sourced by the 100 MHz FPGA clock of the Zynq system-on-chip, which results in a resolution of 10 ns per clock tick. The counter value is provided as input for the GPIO monitoring and power profiling cores. Since this clock domain is local to the FPGA, we have to convert the counter value into a global UTC timestamp before forwarding the data to the testbed controller. In the following, we describe two methods to perform this conversion from local clock values to UTC timestamps. The first option is to use the pulse-per-second (PPS) signal of a GPS receiver, which provides a highly accurate time pulse synchronized to the start of each UTC second. For our testbed deployment, we use a GPS module based on the u-blox MAX-8 receiver, which provides a synchronization accuracy of less than 60 nanoseconds to the GPS time. Thereby, the rising edge of the PPS signal is captured using the GPIO monitoring module, which serves as a reference point between the local 100 MHz counter value and UTC time. Using linear regression, we can estimate the clock drift and offset between the two time domains, which allows to convert subsequent counter values into UTC timestamps used by GPS monitoring or power profiling modules. Since GPS-based time synchronization can be achieved with even a low number of visible satellites, it is enough to place the GPS antenna close to a window or use an active GPS antenna with an extension cable. In cases where observers have to be deployed where GPS signals are not available, the Linux system clock of the observer can be synchronized to UTC using the NTP or PTP time synchronization protocols, which only require connectivity to a clock server. By generating a software-triggered GPIO event based on the system clock, we can then establish a reference point between the 100 MHz counter and the system clock.

Virtual Sensors and Actuators. With the VIADUCT architecture, we introduce the concept of virtual sensors and actuators to facilitate working with a virtual physical environment during testing. The key idea is to utilize a virtual sensor or actuator that has the same electrical interface to the target device as the real hardware component. Consequently, the target device is able to execute the same binary image as it would use with the actual hardware. This is important, as we will keep the same timing behavior of the original application when interacting with the virtual hardware. In this paper, we present a proof-of-concept implementation using the I2C interface, which is widely used to interface external sensors and actuators with microcontrollers. However, the concept presented here is generic and therefore applicable to other electrical interfaces, such as SPI, 1-wire, I2S, UART or GPIO signals.

Each virtual sensor or actuator is implemented as a combination of logic cores and software modules, as depicted in Fig. 6. In the example of an I2C-based peripheral, we use a I2C slave logic core from Xilinx and connect it to a MicroBlaze soft-core processor deployed to the FPGA fabric. The MicroBlaze processor handles low-level read/write requests from the I2C master and emulates the command set and register space of the original peripheral. When implementing the functionality of the virtual peripheral using standard C programming language on the MicroBlaze processor, care has to be taken to properly mimic the original behavior of the device. Both the soft-core processor and the companion software module implemented in Python running on the ARM processor have access to a shared block memory in the FPGA, which allows to exchange state between the two domains. Signaling of events between the ARM and soft-core processor is implemented using interrupt signals. Upon events signaled by the soft-core processor, the Python-based software module will update the local model of the environment. For example, updating the state of a virtual actuator will trigger an update to the state of a virtual sensor and vice versa. Furthermore, the software module will publish events to the controller in order to update the global environment state if necessary and subscribes to updates sent by the controller.

Our proposed approach provides a unified framework to implement various classes of virtual actuator and sensor devices based on the application requirements. The use of the MicroBlaze soft-core processor programmed using the C language and the companion Python module in user-space facilitates integration of new virtual sensors and actuators for users with no prior FPGA development experience.

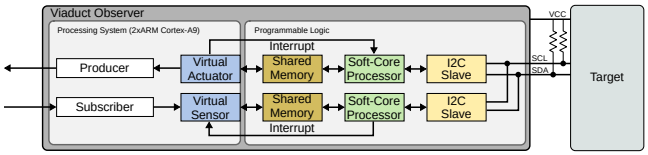


Figure 6: Virtual sensors and actuators combine hardware, logic cores and software components to emulate the virtual physical environment of the target under test.

5.2 Message Broker

The message broker is a key component of VIADUCT's architecture and takes care of message routing between different system components. We employ RabbitMQ[1], which is a popular open-source message broker implementing multiple messaging protocols, such as for example the Advanced Message Queueing Protocol (AMQP). Client libraries for RabbitMQ are available for many different programming languages, which supports flexibility to implement software modules using different programming languages and facilitates future extensions.

Exchanges and Queues. Messages published to the broker are delivered to the designated *Exchange*, which is delivering

[1] https://www.rabbitmq.com

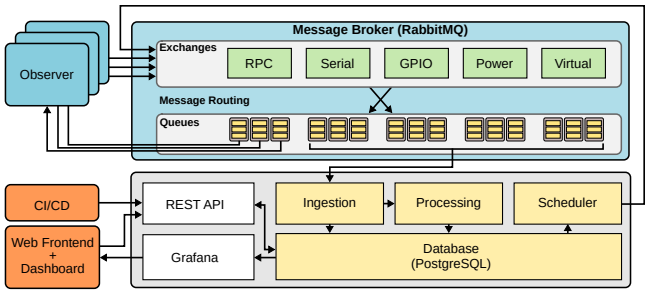


Figure 7: Message broker (RabbitMQ) and core components of the VIADUCT controller.

the message into one or several *Queues* based on the specified routing rules. RabbitMQ allows to route messages into queues based on fields in the *message header* or based on a *routing key* provided together with the message. So called *fanout* exchanges further allow to distribute a message, such as a control command, to all connected subscribers. Software components deployed as part of testbed observers or the controller connect as a subscriber to a particular queue and/or publish messages to exchanges.

5.3 VIADUCT Controller

The roles of the testbed controller are manifold:

- Orchestration of testbed experiments by programming the targets, setup and teardown of tests.
- Collection and storage of measurement data.
- Stream and post-processing of measurements into meaningful results (*e.g.*, energy consumption, analysis of serial output).

Containers. The VIADUCT controller and message broker are implemented as a collection of the following Docker containers running on Linux, as shown in Fig. 7:

- REST API and webserver based on the Flask web application framework (Python)
- Message broker (RabbitMQ)
- Database (PostgreSQL)
- Job scheduler (Python)
- Data ingestion and processing modules (Python)
- Dashboard for visualization (Grafana)
- Monitoring of the observer status (Prometheus)

Utilizing a micro-services architecture based on Docker containers provides flexibility when deploying the controller to different host machine classes (*e.g.*, virtual machines, workstations, servers) and allows to scale up when needed.

We offer two methods how users or other systems can interact with the testbed. First, users can utilize the web interface to submit a testbed job including binaries for the targets. Alternatively, the underlying REST API used by the web interface can also be accessed directly by scripts or other systems, *e.g.*, continuous integration frameworks, to submit jobs and retrieve the measurement data and processed results. Furthermore, the current operational state of

the testbed and information about observers and target devices can be obtained from the REST API.

Test Execution. Submitting a testbed job using the web frontend or the REST API will insert a new job into the database. The job scheduler will execute pending jobs in a first-come first-server manner. First, the binary images are distributed to the observers participating in the test and are then programmed to the target's flash memory. Next, the software modules on the observers are initialized for logging serial, tracing GPIO and/or power profiling as requested. All targets are kept in reset state until the synchronized starting point of the test. After the specified test duration has elapsed, the targets are stopped and cleanup phase is performed.

Data Processing. Measurement data from the observers are continually streamed to the message broker during the test execution phase. The *ingestion* module on the controller will de-serialized the protocol buffers and insert into the data into the corresponding database tables. Stream processing allows to aggregate the data on-the-fly, for example, in order to obtain an up-to-date estimate of energy consumption or count certain GPIO events. In addition, we run several post-processing scripts upon completion of each test with access to all collected measurements, which can be used to perform verification of integration tests or similar. Both raw measurement data and processed results are available to the user or other systems through the REST API.

6 Experimental Evaluation

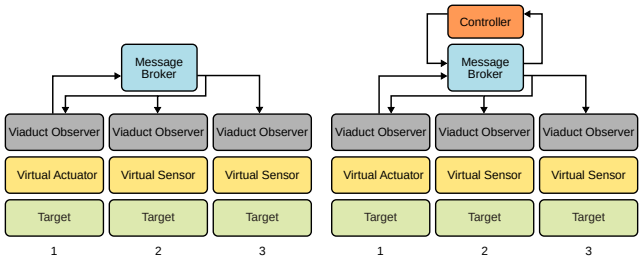
In this section, we present the results of an experimental evaluation of VIADUCT in action and show that the implementation meets our requirements as demonstrated by several micro-benchmarks.

Testbed Setup. Our prototype implementation of the VIADUCT testbed architecture consists of 16 observer nodes based on the Digilent Arty-Z7 board with the nRF52840-DK board as our target mounted inside a 26x18 cm plastic enclosure (see Fig. 1). We have deployed the 16 observers across two buildings and several floors. All observers are connected to the local area network using Ethernet cables. The testbed controller is running on a dedicated server machine connected to the same network. Clock synchronization is achieved by connecting the PPS output of a Mikroe GNSS 5 Click module, which features a u-blox NEO-M8N GPS receiver, to each observer, as described in Sec. 4.

6.1 Latency

In this micro-benchmarking experiment, we evaluate the latency introduced by software, hardware and networking components when collecting and distributing status updates between the broker, controller and observers. Achieving low-latency for status updates will ensure that observers and the testbed controller have a consistent view of a shared global state variable.

Experiment Setup. Two separate test cases are considered to measure latency, as shown in Fig. 8. In Test A, Target 1 generates an actuation event by modifying a status value, which is then sent by Observer 1 to the message broker and then delivered to Observers 2 and 3. In Test B, an actuation event generated by Target 1 is sent through the message broker to the testbed controller. After updating the global



(a) Test A: Obs. 1 → Obs. 2/3

(b) Test B: Obs. 1 → Controller → Obs. 2/3

Figure 8: Testbed setup and message routing for latency measurements between virtual actuators and virtual sensors.

model state at the controller, the updated status value is delivered to all observers. We timestamp all messages sent to the RabbitMQ broker at the source and destination using the synchronized UTC time and calculate the latency as the difference between the two timestamps. Both broker and observers are connected to a corporate local area network infrastructure which is used by other devices and services at the same time.

Results. Fig. 9 shows the distribution of the latency measured by sending 3,000 messages with an interval of 1 second for each test case. For test case A, we observed a mean round-trip time of 5.30 ms (Obs. 1→ Obs. 2) and 5.37 ms (Obs. 1→ Obs. 3), respectively. For test case B, where the message has to pass through the controller first in order to update a global state variable, the mean round-trip time measured increased to 7.38 ms (Obs. 1→ Controller→ Obs. 2) and 7.37 ms (Obs. 1→ Controller→ Obs. 3), respectively. The 95th percentiles of the round-trip time are 7.14 ms and 7.21 ms for test case A, and 9.25 ms and 9.21 ms for test case B. Consequently, our testbed architecture enables us to propagate changes in global state variables within less than 10 milliseconds in most cases, which is sufficient for many CPS applications, *e.g.*, condition monitoring, where sensors and actuators are polled/updated by the target device at a comparable or longer time interval. Furthermore, accurate time synchronization of all observers provided by VIADUCT allows to update the state of virtual sensors precisely at a predefined time in the future, thereby not being affected by jitter due to communication latency.

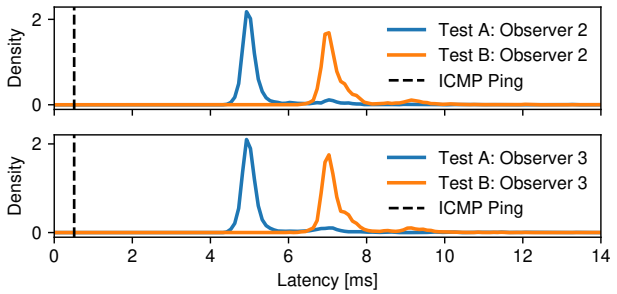


Figure 9: Latency measured between Observer 1→2 and Observer 1→3 for both test cases. The dashed line indicates the average round trip latency measured by ICMP pings.

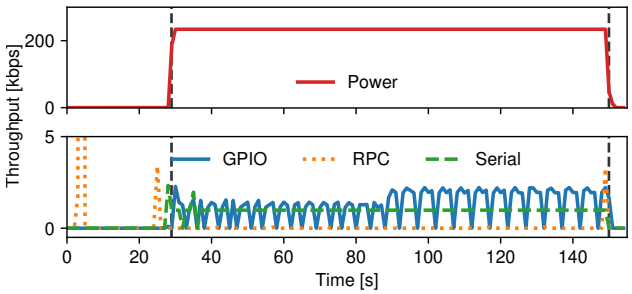


Figure 10: Throughput of AMQP connections between observer and controller measured during setup, execution and teardown of a testbed job. The target is started at $t = 29$s and is halted at $t = 150$s, as indicated by the dashed vertical lines.

6.2 Throughput

Long-time experiments with CPS and near real-time monitoring of testbed jobs require that measurement data and events generated during an experiment can be delivered continuously and with low-latency to the testbed controller. We demonstrate that streaming measurement data from a large number of observers is feasible with the available bandwidth of state-of-the art local area networks. Fig. 10 shows the throughput of incoming and outgoing AMQP connections between a single observer and the RabbitMQ message broker during a short testbed experiment.

Experiment Setup. A job is scheduled for execution on the testbed at $t = 0$s, which triggers a series of RPC commands from the controller to the observer in order to program the binary image and prepare the observer for the test. Upon completion of programming, the test is started at $t = 29$s by releasing the target's reset line. The test is completed by asserting the reset line of the target at around $t = 150$s. It can be observed that streaming the samples from the power profiler collected at 1,000 samples per second results in a throughput of approximately 234 kbps. All collected samples are quickly delivered and throughput drops sharply after the end of the test. Serial logging contributes to around 1 kbps only, as the test applications outputs only a few characters once every second during the testing interval. The throughput of GPIO events increases from 0.9 kbps to roughly 1.6 kbps, as we have configured GPIO actuation by toggling an input pin of the target every 200 ms starting from about 60 seconds after application startup ($t = 89$s).

Results. Based on our measurement results we argue that today's local area and wide area network architectures provide the necessary bandwidth for control traffic and collection of measurement data for a large number of observers. The bandwidth requirements per observer are in the order of 250 kps, which allows to operate 20 observers at a bandwidth of 5 Megabits per second, which is equivalent to a single high quality video stream.

6.3 Virtual Sensors

In this benchmark, we demonstrate that VIADUCT's virtual sensor approach enables testing of CPS applications without having the actual hardware connected to the device under test. Furthermore, no modifications to the binary application running on the target are required, as the functionality of the virtual sensor is emulated at the electrical interface level, as described in Sec. 4.

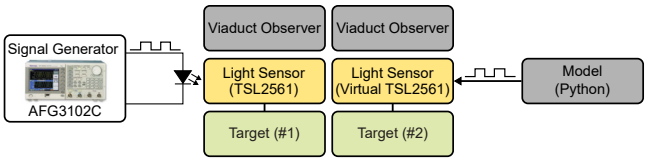


Figure 11: Target 1 is connected to a TSL2561 I2C light sensor to measure the light emitted by the bi-color LED, which is toggled by the AFG3102C signal generator (left). The observer provides a virtual TSL2561 I2C light sensor connected to Target 2 (right).

Experiment Setup. We perform an experiment with two observers connected to our testbed infrastructure. Observer 1 is connected to the nRF52840 target with a TSL2561 ambient dual-channel light sensor attached as a slave to the I2C bus. While the same target device is used with Observer 2, we have not connected an external light sensor, but we connected a virtual light sensor to the I2C bus instead. This can be achieved by connecting the I2C bus of the Nordic nRF52840 to the FPGA fabric of the Zynq-7020, which serves as a I2C slave in this configuration. Please note that hardware-based sensors and virtual peripherals can be combined on the same I2C bus, as long as they have different I2C addresses.

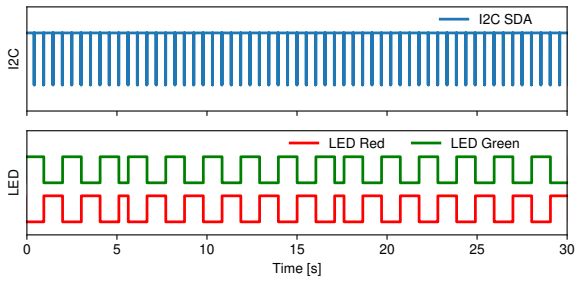
At the start of the testbed job, the same binary application is programmed to both target devices. While the application is running on the target, every 500 ms it will read both channels of the light sensor and convert the result into the corresponding Lux value. Based on a threshold light value, we then either switch on the red or green LED of the target.

Results. In the case of Observer 1, we connect a Tektronix AFG3102C arbitrary waveform generator to a bi-color LED and alternate between two discrete voltage levels at 1 Hz, which will emit red or green light. On the other hand, Observer 2 is connected to a virtual light sensor to mimic the exactly same behavior in software. Therefore, we update the light value registers of the virtual TSL2561 sensor with the corresponding values once every second by our custom Python module. GPIO traces collected during the experiment of the I2C SDA signal and the GPIO signals to the red and green LEDs are depicted in Fig. 12, which confirms that the test application running on the target exhibits the similar behavior regardless of the differences between the hardware-based and virtual light sensors.

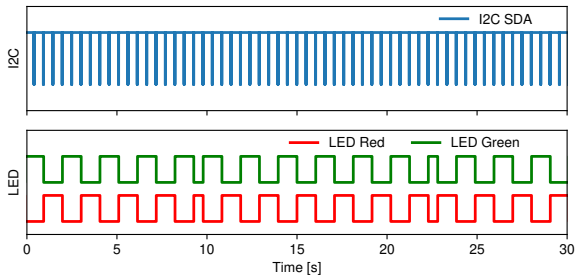
7 VIADUCT in Action

This section demonstrates how VIADUCT can be used to evaluate the behavior of cyber-physical systems that interact with complex and spatially distributed physical processes.

Use Case Description. We consider a use case central to modern smart building applications, namely, the localization of occupants within a building using acoustic analysis. In order to achieve this, we assume that each room of the smart building is equipped with wireless sensor nodes that are ca-



(a) Observer 1 with TSL2561 light sensor.



(b) Observer 2 with virtual TSL2561 light sensor.

Figure 12: Timeline of GPIO traces of the I2C data line and LEDs during experiments with a TSL2561 light sensor (top) and virtual light sensor (bottom). The bi-color light sources are toggled every second between red and green light.

pable to synchronize to a global time base over a wireless network. Each sensor node is equipped with a microphone to periodically collect audio samples and executes a classification algorithm to determine if an acoustic event (*e.g.*, an alarm sound) is detected. If an event is detected, the sensor node will generate an event containing a timestamp of the global time base and will proceed to disseminate the event through the wireless network to a gateway. If the gateway receives many such events from different wireless nodes, it can calculate the approximate location of the acoustic emission by comparing the timestamp encoded in each event.

Challenges of Testing. In a real-world deployment, such a system is very difficult to evaluate. The key challenge is that the underlying physical process, *i.e.*, many acoustic signals from many occupants, becomes infeasible to evaluate as the number of wireless nodes in the network increases. For example, if there is only one sensor node, a signal generator can be connected to the microphone device to playback representative acoustic signals. However, as the number of nodes increases, additional signal generators are needed to represent attenuation, echoing and overhearing characteristics associated with a real-world acoustic channel, thus making such an approach financially infeasible and complex to implement.

Testing with VIADUCT. The VIADUCT testbed infrastructure transforms this infeasible testing problem into a flexible and scalable solution by modeling the behavior of the complex acoustic channel as a virtual sensor, *i.e.*, a *virtual microphone*. Specifically, the virtual microphone generates a stream of acoustic samples, *i.e.*, encoded as I2S digital signals, that are time synchronized to the global time base

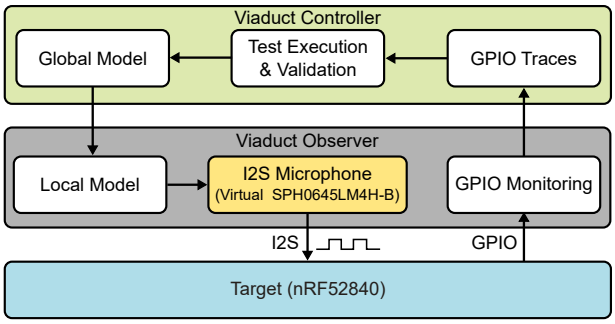
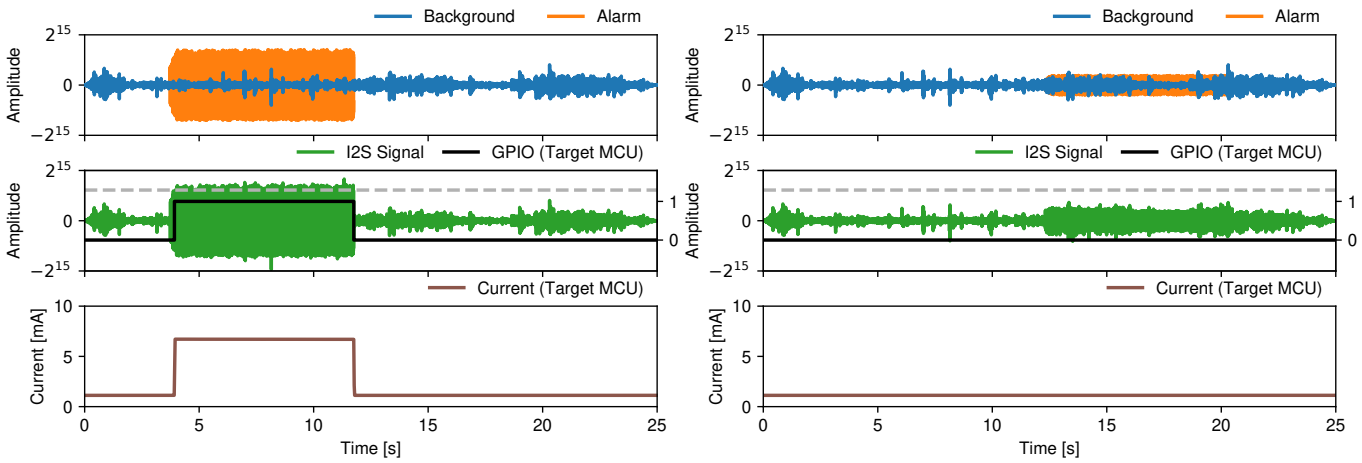


Figure 13: Experiment setup: A virtual microphone is connected to the target node using the I2S protocol. The target MCU's GPIO events are captured by the observer.

maintained by each VIADUCT observer. When the sensor node detects an event of interest in the acoustic signal, a GPIO line is triggered, and the VIADUCT observer records a global timestamp of the event. Attenuation and echoing characteristics of the acoustic channel may be adjusted at each VIADUCT observer instance in real-time by appropriately manipulating the audio stream controlled by the local physical model of the audio channel. The overhearing characteristic of the acoustic channel may also be reproduced since all VIADUCT observes are synchronized to the same global time base. The global physical model is implemented as a Python module running in the VIADUCT controller, which publishes location and intensity of sound events to the local model instances running on the observers. Upon reception of a new sound event request, the local model overlays the corresponding audio file onto the ambient noise and transmits the resulting audio segments using the I2S stream.

Experimental Setup. For this use case, we employ a virtual microphone, which emulates a I2S microphone sensor (*e.g.*, the Knowles SPH0645LM4H-B), by instantiating a I2S transmitter core in the programmable logic of the Zynq system-on-chip. Using the ALSA subsystem, the I2S transmitter is made available as a regular sound device in Linux. Transmitting an I2S output stream can be initiated by writing the corresponding audio samples encoded with Pulse-code modulation (PCM) to the sound device. The I2S signal lines are passed through a level shifter and connected to the I2S input pins of the nRF52840 microcontroller. The setup of the experiment is depicted in Figure 13. We implemented a test application running on the nRF52840 that is continuously sampling the I2S input and writes the data to one of several memory buffers. Once a buffer is full, we execute a peak detection algorithm on the audio samples to determine if the acoustic signal exceeds a defined threshold in order to distinguish from ambient noise. To demonstrate the use of the virtual microphone for detection of acoustic events, we trigger a GPIO output pin, which is monitored by the observer. Implementation of further signal processing steps and sound classification algorithms (*e.g.*, based on machine learning methods) are beyond the scope of this case study. However, we emulate the increase in power consumption of the MCU due to the additional processing cycles required for sound classification by busy waiting.



(a) High amplitude alarm signal starting at 4 seconds.

(b) Low amplitude alarm signal starting at 12 seconds.

Figure 14: Experiment with VIADUCT's virtual microphone sensor: An alarm signal at high (left) or low amplitude (right) is mixed onto the background noise. The resulting sound signal is transmitted to the target node using the I2S interface to emulate a microphone sensor. As the signal exceeds the threshold value for the high amplitude alarm, the application running on the target MCU detects the alarm signal and starts the classification algorithm. As the amplitude of the mixed signal is below the threshold for a low amplitude alarm, the classification algorithm is not activated by the signal.

Results. We performed two test cases using the virtual sensor capabilities of the VIADUCT testbed. In the first experiment, we generate an acoustic alarm signal at high amplitude and add it on top of a sound file representing common office noise. The superposition of the two signals is then sent using the virtual I2S microphone to the target node. In the second case, we keep the original background noise, but add an alarm signal with a lower amplitude at a different position in time. In both cases, the application binary executed on the target MCU will continually sample the incoming I2S data and perform peak detection by comparing the maximum value within a data block with a predefined threshold value. It can be observed that sampling the I2S data and periodically checking the acquired samples against the threshold value will result in a current drain of roughly 1 mA, as measured by the VIADUCT's power profiler. In case the threshold value is exceeded during the current audio block, a GPIO output pin is set to logic high, otherwise it is set to logic low. We report the sound samples generated by the virtual microphone as well as power and GPIO traces from the two experiments in Figure 14. As expected, the target node has detected the high amplitude alarm signal in the first test case, as indicated by the signal level of the output pin. Furthermore, we can observe that the current drawn by the target increased to roughly 6-7 mA, which can be attributed to the busy wait operation. In the second test case, the detection threshold for the sound input signal has not been exceeded, as indicated by the output pin remaining low. While existing testbed infrastructures provide capabilities such as serial port logging, GPIO monitoring and power profiling, VIADUCT provides in addition to that also fine-grained control of the sensor input to the target nodes during a testbed experiment and allows to reproduce the exact same test scenarios when validating different implementations of the target application.

8 Discussion

We next discuss some limitations of VIADUCT and possible future improvements.

Time Synchronization. The network time synchronization used by VIADUCT is based on GPS synchronization. Since GPS has severe performance limitations in an indoor environment, which is where a VIADUCT testbed would typically be installed, *i.e.*, in a laboratory of industrial environment, one could instead employ the IEEE 1588 Precision Time Protocol (PTP) over Ethernet. Alternatively, a custom time synchronization protocol based on constructive wireless interference, *e.g.*, such as the Glossy [13] primitive, may be used for sub-microsecond time synchronization between the VIADUCT controller and observers, as shown in [23].

Power Profiler. The voltage and current measurement circuits used in VIADUCT are extremely sensitive, and due to poor ESD protection, there exists a small 50 Hz interferer. This may be mitigated using appropriate ESD materials, *e.g.*, ferromagnetic shielding between the measurement circuits and the power supply. It is duly noted that the front-end analog circuits for voltage and current measurement are susceptible to temperature variation of the passive components, however, this has been minimized through the use of low-tolerance components and appropriate calibration.

Extensibility. The VIADUCT testbed presented in this work does not emulate environmental conditions that may be expected in a harsh environment, which is typically where cyber-physical systems for industrial applications would be deployed. For example, the adverse effects of temperature variation and wireless interference are well known, as detailed in [6] and [3], respectively. However, the flexibility of the VIADUCT architecture enables the extension of features that can emulate such environmental conditions, for example through integrating suitable components from the TempLab [9] and JamLab [8] testbeds.

Outlook. Despite the aforementioned limitations, we believe that the VIADUCT testbed architecture is an important step toward addressing the challenges of dependable cyber-physical systems, as recently surveyed in [14]. Since the costs and difficulty associated with troubleshooting cyber-physical systems in the field are extreme, as exemplified in [33], there is a need for new methods and tools such as VIADUCT, to ensure the complex interactions of a real-world deployment are exposed to the cyber-physical system prior to deployment. The unique characteristics of VIADUCT make it possible to quantify metrics of dependability that better resemble a real-world deployment, and thus move one step closer to realizing dependable systems in practice.

9 Conclusion

We have argued that state of the art testbeds are not sufficient to effectively evaluate cyber-physical systems, since they do not capture the complex reactive requirements imposed by real-world physical processes. We therefore propose, VIADUCT, a scalable and high-precision testbed architecture that abstracts the underlying physical process of a cyber-physical system and thus making it possible to inject stimulus comparable to a real-world deployment.

In this paper, we have presented the design and implementation of a 16-node VIADUCT testbed based on well-established system design principles. We experimentally evaluated the VIADUCT testbed using several case studies. The results have shown that the proposed VIADUCT architecture supports real-world reactive requirement evaluation through the virtual sensor and actuator abstraction and scales well with the size of the cyber-physical system under test.

10 Acknowledgments

The authors would like to thank Franz Zurfluh for his valuable contributions to the FPGA implementation.

11 References

[1] C. Adjih, E. Baccelli, E. Fleury, G. Harter, N. Mitton, T. Noel, R. Pissard-Gibollet, F. Saint-Marcel, G. Schreiner, J. Vandaele, et al. FIT IoT-LAB: A large scale open experimental IoT testbed. In *WF-IoT*, pages 459–464. IEEE, 2015.

[2] P. Appavoo, E. K. William, M. C. Chan, and M. Mohammad. Indriya2: A Heterogeneous Wireless Sensor Network (WSN) Testbed. In *Conf. on Testbeds and Research Infrastructures*, pages 3–19. Springer, 2018.

[3] N. Baccour, D. Puccinelli, T. Voigt, A. Koubaa, C. Noda, H. Fotouhi, M. Alves, H. Youssef, M. A. Zuniga, C. A. Boano, et al. External radio interference. In *Radio Link Quality Estimation in Low-Power Wireless Networks*, pages 21–63. Springer, 2013.

[4] D. Baumann, F. Mager, H. Singh, M. Zimmerling, and S. Trimpe. Evaluating low-power wireless cyber-physical systems. In *CPSBench*, pages 13–18, 2018.

[5] C. A. Boano, S. Duquennoy, A. Förster, O. Gnawali, R. Jacob, H.-S. Kim, O. Landsiedel, R. Marfievici, L. Mottola, G. P. Picco, et al. IoTBench: Towards a benchmark for low-power wireless networking. In *CPSBench*, pages 36–41. IEEE, 2018.

[6] C. A. Boano, K. Römer, and N. Tsiftes. Mitigating the adverse effects of temperature on low-power wireless protocols. In *MASS*, pages 336–344, 2014.

[7] C. A. Boano, M. Schuß, and K. Römer. Ewsn dependability competition: Experiences and lessons learned. *IEEE Internet of Things Newsletter*, 2017.

[8] C. A. Boano, T. Voigt, C. Noda, K. Römer, and M. Zúñiga. Jamlab: Augmenting sensornet testbeds with realistic and controlled interference generation. In *IPSN*, pages 175–186. IEEE, 2011.

[9] C. A. Boano, M. Zúñiga, J. Brown, U. Roedig, C. Keppitiyagama, and K. Römer. Templab: A testbed infrastructure to study the impact of temperature on wireless sensor networks. In *IPSN*, pages 95–106. IEEE, 2014.

[10] M. Doddavenkatappa, M. C. Chan, and A. L. Ananda. Indriya: A low-cost, 3d wireless sensor network testbed. In *Intl. Conf. on Testbeds and Research Infrastructures*, pages 302–316. Springer, 2011.

[11] S. Duquennoy, O. Landsiedel, C. A. Boano, M. Zimmerling, J. Beutel, M. C. Chan, O. Gnawali, M. Mohammad, L. Mottola, L. Thiele, et al. A benchmark for low-power wireless networking. In *SenSys*, pages 332–333, 2016.

[12] E. Ertin, A. Arora, R. Ramnath, V. Naik, S. Bapat, V. Kulathumani, M. Sridharan, H. Zhang, H. Cao, and M. Nesterenko. Kansei: a testbed for sensing at scale. In *IPSN*, pages 399–406, 2006.

[13] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *IPSN*, pages 73–84, 2011.

[14] F. Foukalas, P. Pop, F. Theoleyre, C. A. Boano, and C. Buratti. Dependable Wireless Industrial IoT Networks: Recent Advances and Open Challenges. In *European Test Symposium (ETS)*, pages 1–10, 2019.

[15] Y. Gao, J. Zhang, G. Guan, and W. Dong. LinkLab: A Scalable and Heterogeneous Testbed for Remotely Developing and Experimenting IoT Applications. In *IoTDI*, pages 176–188, 2020.

[16] K. Geissdoerfer, M. Chwalisz, and M. Zimmerling. Shepherd: A Portable Testbed for the Batteryless IoT. In *SenSys*, 2019.

[17] V. Handziski, A. Köpke, A. Willig, and A. Wolisz. TWIST: a scalable and reconfigurable testbed for wireless indoor experiments with sensor networks. In *REALMAN*, pages 63–70. ACM, 2006.

[18] T. A. Henzinger. Two challenges in embedded systems design: predictability and robustness. *Philos. Trans. R. Soc. A*, 366(1881):3727–3736, 2008.

[19] T. A. Henzinger and J. Sifakis. The discipline of embedded systems design. *Computer*, 40(10), 2007.

[20] H. Kopetz. The complexity challenge in embedded system design. In *ISORC*, pages 3–12, 2008.

[21] E. A. Lee. Cyber physical systems: Design challenges. In *ISORC*, pages 363–369, 2008.

[22] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. FlockLab: A testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *IPSN*, 2013.

[23] R. Lim, B. Maag, B. Dissler, J. Beutel, and L. Thiele. A Testbed for Fine-Grained Tracing of Time Sensitive Behavior in Wireless Sensor Networks. In *Local Computer Networks Workshops*, 2015.

[24] J. Ma, J. Wang, and T. Zhang. A Survey of Recent Achievements for Wireless Sensor Networks Testbeds. In *CyberC*, pages 378–381. IEEE, 2017.

[25] J. Munoz, F. Rincon, T. Chang, X. Vilajosana, B. Vermeulen, T. Walcarius, W. Van de Meerssche, and T. Watteyne. OpenTestBed: Poor Man's IoT Testbed. 2019.

[26] A. Platzer. Logic & proofs for cyber-physical systems. In *Intl. Joint Conference on Automated Reasoning*, pages 15–21. Springer, 2016.

[27] M. Schuß, C. A. Boano, M. Weber, and K. Roemer. A competition to push the dependability of low-power wireless protocols to the edge. In *EWSN*, pages 54–65, 2017.

[28] M. Schuß, C. A. Boano, M. Weber, M. Schulz, M. Hollick, and K. Römer. JamLab-NG: Benchmarking Low-Power Wireless Protocols under Controllable and Repeatable Wi-Fi Interference. In *EWSN*, pages 83–94, 2019.

[29] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. RocketLogger - Mobile Power Logger for Prototyping IoT Devices. In *SenSys*, 2016.

[30] L. Sigrist, A. Gomez, R. Lim, S. Lippuner, M. Leubin, and L. Thiele. Measurement and Validation of Energy Harvesting IoT Devices. In *DATE*, pages 1159–1164. IEEE, 2017.

[31] P. Sommer and B. Kusy. Minerva: Distributed tracing and debugging in wireless sensor networks. In *SenSys*. ACM, 2013.

[32] G. Werner-Allen, P. Swieskowski, and M. Welsh. Motelab: A wireless sensor network testbed. In *IPSN*, page 68. IEEE, 2005.

[33] U. Wetzker, I. Splitt, M. Zimmerling, C. A. Boano, and K. Römer. Troubleshooting wireless coexistence problems in the industrial internet of things. In *CSE, EUC, and DCABES*, pages 98–98, 2016.