A Testbed for Fine-Grained Tracing of Time Sensitive Behavior in Wireless Sensor Networks

Roman Lim, Balz Maag, Benjamin Dissler, Jan Beutel, Lothar Thiele
Computer Engineering and Networks Laboratory
ETH Zurich, Switzerland
lim@tik.ee.ethz.ch

Abstract—**This paper introduces TRACELAB, a new testbed architecture that allows for fine-grained tracing of time sensitive behavior of low-power wireless embedded systems. Such traces help to systematically analyze code execution to find software errors, measure bounds for execution times, or to verify functional program properties. TRACELAB builds on the idea of GPIO tracing: by including short GPIO instructions into node applications, the program behavior can be traced in a minimally invasive manner, simultaneously on all observed nodes. TRACELAB enables fine-grained distributed tracing by overcoming the limits of existing testbed architectures with respect to timing accuracy and peak event rates. For that purpose, an existing testbed design is extended with a new data acquisition system that includes an FPGA chip for fast and deterministic data handling. To faithfully align distributed trace measurements, TRACELAB integrates a highly accurate wireless time distribution network. We build 31 TRACELAB observers and deploy them in an office environment and outdoors. Measurements using GPS precision timing show that TRACELAB (*i*) is able to trace events at rates of up to 10^8 events/s and (*ii*) aligns traces from different locations within 1 μs with an empirical probability of 99.9 %.**

I. INTRODUCTION

Testbeds have become a fundamental part in the development cycle of wireless embedded systems. Testing a system in a real distributed environment exposes it to external factors like multi-path signal propagation, signal attenuation, temperature changes, or hardware variations. As it is difficult to accurately model and simulate these influencing factors, testing on real hardware is needed for proper evaluation and validation. Testbeds provide engineers and researchers the services to facilitate testing for wireless embedded systems. Typical services are programming, logging of serial port output, and power profiling.

Motivation. While the currently available testbed services already cover many areas beyond the basic needs of wireless embedded application engineers, support still lacks for network-wide, fine-grained cycle-accurate tracing of time sensitive system behavior. To illustrate this, we consider two examples:

1) Low-power MAC protocols require exact and coordinated timing of actions to ensure efficient operation, *e.g.*, Glossy relies on constructive interference of concurrently transmitted packets, which requires transmissions of neighboring nodes to be aligned within 0.5 μs [1]. To observe and validate the interaction between different nodes in such a network, a distributed tracing mechanism must be minimally invasive and deliver the recorded trace of each individual node tightly time-synchronized with all other concurrent traces.

2) Control-flow tracing of programs allows for efficient debugging and to find potential failure causes [2]. In this case, every branch instruction in a program needs to be traced. The resulting enormous volume of tracing points necessitates an efficient and minimally intrusive trace recording system.

Existing approaches are either too intrusive (*e.g.*, using `printf`), or not general enough to meet the diversity of available node platforms (in system debugging [3],[4]). More crucially, none of the available approaches is able to align traces with the required accuracy and cope with high peak event rates.

Contributions. To overcome the limitations of current testbeds, we introduce TRACELAB, a new distributed data acquisition system that is capable of tracing mote application behavior at a high time resolution in a minimally invasive manner, tightly synchronized throughout a network of 31 nodes. The basis of our new system is the FlockLab testbed architecture [5], which provides a distributed network of *observer* platforms that are used to stimulate and monitor the attached devices under test, the *targets*. We build on the idea of the existing GPIO tracing and actuation services, and extend their capabilities with respect to *sampling resolution*, *peak sampling rate*, and *time alignment of traces* by several orders of magnitude. By including short GPIO instructions into node applications, the program behavior can be traced in a minimally invasive manner. The design of the new data acquisition system consists of a field-programmable gate array (FPGA) and a CC430 SoC with RF core. The FPGA chip handles the timing sensitive data acquisition part, while the SoC is running a wireless time synchronization protocol to keep the system time of the FPGA chip on each observer synchronized. TRACELAB requires synchronization accuracy that rules out the commonly used network time protocol NTP [6]. For local networks, the precision time protocol PTP [7] is a more accurate alternative, while GPS receivers provide accurate synchronization on a global scale. PTP requires special hardware support within the network infrastructure, while GPS receivers only provide accurate synchronization in places with good satellite reception. As we want to support both, indoor locations with possibly bad GPS satellite reception as well as outdoor locations with limited infrastructure support, we identify synchronization using a low-power wireless multi-hop network as viable solution. We integrate a synchronization algorithm based on Glossy [1] into TraceLab's data acquisition system, and show that the synchronization performance can be considerably improved by applying a jitter reduction filter.

Challenges. Depending on the node application, tracing an execution path with many conditional branches within a short time window requires an efficient data handling mechanism that can manage high peak data rates. To faithfully capture the interaction between different nodes in the network, measurements need to be accurately time-synchronized. The required combination of high peak sampling rate and accurate time synchronization makes designing such a data acquisition system a challenging task.

Findings. Our system is designed to capture GPIO events with a time resolution of $0.5\,\mu$s. According to the evaluation in Sec. VI, our prototype sustains a peak event rate of 10^8 events/s, while the maximal average event rate the system can handle is 2.85×10^5 events/s. Expressed in numbers of the widely used TelosB node platform, this event rate allows to continuously trace a program of which one third of the instructions change GPIO states.[1] Typical low-power applications exhibit even less events to trace due to energy saving strategies that put the CPU to a sleep mode. Measurements in a 31-node network assisted by GPS precision timing show that TraceLab aligns concurrently recorded traces within $1\,\mu$s with an empirical probability of 99.9 %.

In the following, we discuss related work in Sec. II and derive the requirements for a fine-grained, distributed trace recording system in Sec. III. In Sec. IV, we give an overview of the system design and discuss the data acquisition system and the time synchronization mechanism in detail. We explain implementation specific details in Sec. V, evaluate key properties of TraceLab in Sec. VI, and conclude the paper in Sec. VII.

II. Related Work

Related to this paper are hardware and software solutions that allow to trace embedded system behavior, both on a single entity and at network scale. We further discuss work related to techniques applied in our system design, which revolves around time synchronization using a 1-pulse-per-second (1-PPS) signal and time synchronization protocols for low-power wireless embedded systems.

Tracing system behavior can be realized either on the target device itself, or using external hardware. Software solutions instrument program code at branch instructions and use efficient encoding to log control flow traces to flash memory [2]. Another instrumentation approach is pursued by Tardis [8], which rather logs non-deterministic program inputs as a trace for later replaying using a simulator. While software solutions are easily applicable and can provide very accurate information about the state of a node, the required resources on the target for processing and storing the traces render such an approach unsuitable to trace time sensitive behavior. Indeed, experiments with Tardis reveal that the CPU duty cycle of standard node applications can almost double with tracing enabled [8].

Additional hardware offloads data processing from the target to an external observer platform, providing an out-of-band communication channel in addition. Two different data extraction methods are commonly applied in this context: On-chip debug interfaces or simple GPIO pins for binary state information. Aveksha [4] uses a debug board extension to trace events of interest on a single target using the on-chip debug module of the MSP430 microcontroller. A low-cost and networked solution is provided by Minerva [3]. Tracing using on-chip debug interfaces is non-intrusive and expressive, but not easily portable between different microcontroller architectures. Monitoring GPIO pins is a more generic approach, at the expense of slightly higher intrusiveness caused by short GPIO instructions. GPIO pins can be traced at relatively high speeds, as shown in a FPGA-based logic analyzer design that is able to sample 8 GPIO pins on a single embedded system at a rate of 200 events/s [9]. GPIO tracing in a distributed fashion is also a key element of FlockLab [5]. Different to TraceLab, the aforementioned single node monitoring solutions don't provide a consistent global view of a network. Available networked solutions are only conditionally suited for fine-grained distributed trace recordings due to their limited tracing rates and time synchronization accuracy, *e.g.*, FlockLab traces exhibit a maximal pairwise timing error of $255\,\mu$s and contain events at a maximal rate of 3.5 kHz for lossless traces. TraceLab is a distributed tracing solution that overcomes these limitations.

TraceLab implements a digital loop control algorithm to lock the FPGA-internal system clock to a 1-PPS signal. Similar controllers have been used in the past, *e.g.*, to reduce the jitter of a GPS 1-PPS signal [10].

As further detailed in Sec. III, observers in TraceLab have to be time-synchronized within $1\,\mu$s, which we achieve using a multi-hop time synchronization protocol. A popular time synchronization algorithm is employed by the flooding time synchronization protocol FTSP [11]. However, the achieved maximal synchronization error of less than $14\,\mu$s in a 6-hop network does not meet our requirements. Schmid et al. improve on the achieved accuracy of FTSP by introducing a high resolution clock [12]. Glossy, a flooding architecture for wireless sensor networks that exploits constructive interference for fast network flooding, implicitly provides time synchronization [1]. On a TelosB, the reported average error over 8 hops is as low as $0.4\,\mu$s, with a standard deviation of $4.8\,\mu$s. In TraceLab, we port Glossy to a node platform that has two distinct properties that improve time synchronization: (*i*) as in [12], a high resolution system clock, and (*ii*) a radio chip with automatic RX/TX transceiver mode switching. Additionally to the baseline Glossy, we improve on the time synchronization variance by adding a jitter reduction filter.

[1]Assuming a clock speed of 4 MHz.

III. Enabling Fine-Grained Tracing

In this section, we sketch the idea of tracing system behavior using GPIO pins as monitoring interface, and we derive the requirements needed to actually enable fine-grained tracing of system behavior of low-power wireless sensor networks.

In this context, we specify the system behavior of interest as the control path taken during a program execution on a set of nodes, annotated with time information. This information can be used in several ways to analyze code execution. Examples are (*i*) the quantification of code coverage for test applications, (*ii*) empirical determination of bounds for execution times of certain program parts, or (*iii*) the verification of system behavior against a given specification using exhaustive testing.

To get execution traces of programs, we instrument the program code using short *marker instructions* to emit pin state changes (*events*) at branches in the program flow. As these instructions are very short, we assume that these additional instructions only minimally affect the system behavior. We then reconstruct the taken program flow from the emitted sequence of pin level changes. This is feasible if the emitted GPIO trace is ordered by time and unambiguously mappable to a sequence of program executions. Ensuring unambiguous mapping by means of a limited number of pins possibly requires to encode individual markers using a sequence of pin changes.

Table I
Peak event rates for different target platforms.

Node platform	Peak event rate (events/s)	Cycles	Pin change time
TelosB	0.8×10^6	5	1,250 ns
Tinynode 184	2.4×10^6	5	417 ns
IRIS	1.6×10^6	2	250 ns
Opal	19.2×10^6	5	52 ns

What are the requirements for a trace recording system? The rate of the emitted GPIO level changes is limited by the maximal pin setting rate of a target, which in turn depends on the target's MCU clock speed and the number of instructions needed to change a pin state as summarized in Table I. This rate might be reached if programs have several conditional branches in a row, or if markers in the program code are encoded using sequences of pin changes. On the other hand, low-power wireless embedded systems are usually duty-cycled to save energy. There is no code execution during sleep states, and therefore the expected average tracing rate can be significantly lower than the peak event rate. These observations lead to the first requirement:

Requirement 1. A trace recorder needs to sustain for short periods of time a peak sampling rate that is able to capture the maximal pin level change rate of a target.

The time resolution of a trace should be sufficiently high to allow meaningful execution time measurements, thus our second requirement is:

Requirement 2. The trace of a single target must be ordered by time and exhibit a sub-microsecond time resolution.

To observe interactions between several nodes in a network, and to properly order these interactions relative to each other, time annotations within a trace must allow to sufficiently align it with traces of neighboring nodes. For instance, to measure and properly adjust timing properties like wake-up guard times of low-power MAC protocols, an alignment error in the range of the smallest controllable time quantity on a node would be preferable. Behavior is typically controlled by timers running from 32 kHz oscillators or the main system clock, which allows control at a resolution of a few microseconds. We therefore phrase the last requirement as follows:

Requirement 3. The time synchronization error between observers of neighboring nodes should be in the lower microsecond range.

Next, we describe the architecture of TraceLab, and we explain how we address the given requirements.

IV. Architecture

A. Overview

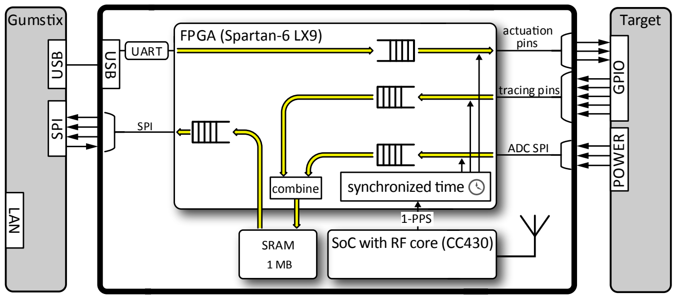

Figure 1. Overview of a single TraceLab observer.

Fig. 1 provides an overview on the system architecture of TraceLab. The newly designed system fits into the existing FlockLab architecture and replaces the existing data acquisition part, which is originally running entirely on a Gumstix embedded computer.

The Gumstix features a 624 MHz Marvell XScale PXA270 microprocessor that runs OpenEmbedded Linux and is equipped with 128 MB SDRAM, 32 MB flash memory and an 8 GB SD card. All observers in TraceLab are connected over Ethernet or Wi-Fi (outdoor) to a backend infrastructure.

We trace a target in TraceLab by means of two different tracing interfaces, GPIO lines and an ADC, the former for digital state information, the later for power measurements. The GPIO interface is used in two directions, either controlled by the target or by the observer. The state of these interfaces are traced by the TraceLab board. Traces are annotated with a timestamp and forwarded to the Gumstix computer. We handle all time critical tasks on the TraceLab board, while test management and communications tasks are allocated to the Gumstix computer.

The TraceLab data acquisition architecture consists of a Spartan-6 FPGA chip, a static random-access memory (SRAM) and a CC430 SoC that combines an MSP430 micro-controller and a CC1101 radio transceiver chip. Functionally, the system has to process three different types of data streams: (*i*) GPIO actuation commands to control 3 GPIO pins, (*ii*) GPIO tracing on 5 pins and (*iii*) power profiling data. In total, 9 individual streams need to be processed in parallel and with low time jitter. We employ an FPGA chip for this task, because the parallel nature of such a chip allows to map the processing of each stream type to dedicated hardware modules, and therefore greatly facilitates a deterministic, low jitter processing. In contrast to this paradigm, many existing testbed architectures, including FlockLab, rely on sequential processing on a single processor.

We tackle the high peak sampling rate requirement by employing a hierarchical memory structure. Fast on-chip FIFO queues within the FPGA handle short bandwidth spikes, while the SRAM memory is used to buffer larger amounts of sampled data, before we finally write the acquired traces to the serial peripheral interface (SPI) bus for further storage on an SD card on the Gumstix.

To put the measured information into a global time context, we need to keep the time on each observer synchronized with all the other observers. For this purpose, the internal time of the FPGA is disciplined by applying a 1-PPS signal. The edge of such a pulse indicates the start of a new second. Typically, GPS receivers provide this kind of pulse for synchronization purposes. However, relying on GPS timing restricts the range of use to locations with good satellite reception, which rules out most indoor locations. To distribute a time pulse to all observers in the testbed, our design relies on a wireless time synchronization protocol based on Glossy [1], running on the SoC of the TRACELAB board.

Next, we describe in Sec. IV-B the clock control algorithm that keeps the internal FPGA-time locked to a 1-PPS signal time. Sec. IV-C details the FPGA-design of our data acquisition system, Sec. IV-D explains the configuration interface, and Sec. IV-E describes how we accurately synchronize time on all observers in the testbed to a common reference.

B. Disciplined System Clock

In the following, we discuss time and clock related details of the FPGA design. The FPGA chip runs at a clock speed of 100 MHz. At the same time, this is also the maximally achievable sampling rate of GPIO states. To facilitate digital control, we derive a *system time* counter, running at a nominal frequency of 2 MHz, that is a clock period of 500 ns. The speed of the system is adjustable by varying the number of FPGA clock cycles per system time clock tick.

To keep the system time in line with the external reference 1-PPS signal, we employ a clock control algorithm that combines an open-loop controller with a feedback control loop, as illustrated in Fig. 2. The open-loop controller determines the local clock rate by averaging the number of FPGA clock cycles within N 1-PPS periods, where $N = 8$ in our design. We denote the difference between this measured clock rate and the nominal clock rate as $e_r(t)$. The feedback control loop implements a P-controller, which corrects the offset of the system time relative to the external 1-PPS signal. The offset $e_o(t)$ is measured using the offset detector in Fig. 2. The sum of $e_r(t)$ and $e_o(t)$, expressed in system clock cycles, is given as input to a variable clock divider that generates the 2 MHz system time clock based on the 100 MHz FPGA system clock. The clock divider samples the system clock down by a variable factor $\beta \in \{49, 50, 51\}$. The factor is applied in a way, such that the total error is corrected evenly spread over the period of one second.

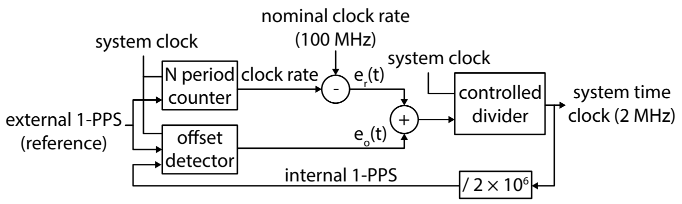

Figure 2. The clock control algorithm is a combination of an open-loop controller and a feedback control loop. The speed of the system time clock is controlled by a variable divider.

We evaluate this design in Sec. VI-B and show that our controller keeps the internal system clock tightly synchronized to the 1-PPS signal.

C. Data Acquisition

The data acquisition part of the FPGA-design captures the data generated by the target, that is state changes of the target's GPIO lines and the power dissipation of the target, measured by the ADC. Each captured element has to be annotated with a precise timestamp and forwarded to the Gumstix computer.

The two data streams that need to be handled have different properties. GPIO state changes, which are triggered by single instructions on the target platform, occur at irregular time intervals and possibly exhibit high peak rates, *e.g.*, an Opal node could potentially emit 19.2×10^6 state changes per second. Power measurements on the other hand follow a strict sampling interval and occupy less bandwidth, in our case up to 56 ksps. Our design takes these differences into consideration by applying different internal data representations. To maximize throughput, each acquired data element should occupy as little memory as possible. Space can be saved by not including the complete timestamp into each data element, but rather stripping the most significant bits from the time value. As depicted in Fig. 3, we introduce *marker elements* (b) and (d) into the data stream to mark the change of the significant bits in the stream. The complete time can be reconstructed in a later step, *e.g.*, on a back-end server. For power profiling samples, the knowledge of the sampling rate even eliminates the need for storing the lower part of the timestamp, *i.e.*, the essential data consists of an ADC sample and a header, as in (e) in Fig. 3. Packet format (c) is needed to indicate the start and end of a power profiling trace, which can happen at arbitrary time instants. While our approach generates a low-rate base stream of metadata, it also greatly reduces the data volume for event bursts and power profiling.

The memory structure matches the input data stream (*i.e.*, GPIO events and the power profiling), to the output, in our

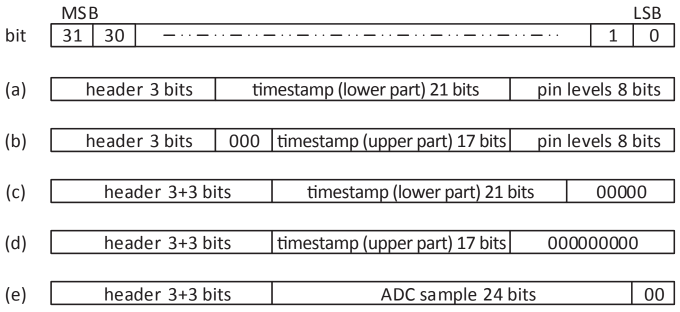

Figure 3. Data format of GPIO tracing and power profiling packets. All packets have a width of 32 bits and include a header of 3 bits. *Marker packets* (b) and (d) contain the upper 17 bits of the timestamp.

case an SPI bus running at a clock frequency of 12 MHz. As illustrated in Fig. 4, the output has a significantly lower peak bandwidth than the input. Therefore we resort to a hierarchical memory structure that provides both, high short term bandwidth on the input and enough buffer space to shape the data stream to match the slower output. Small and fast FIFO queues can handle new data packets with every clock cycle, while the access time to the SRAM amounts to 16 cycles, *i.e.*, a maximal packet rate of 6.25×10^6 packets/s.

The two input streams are merged before writing to the SRAM chip. To avoid starvation, power profiling data is prioritized because there is a tight and not saturating upper bound to the maximal data rate of this stream, which is less than 1 % of the SRAM memory bus bandwidth.

We empirically evaluate the throughput of the data acquisition with the help of an event generator in Sec. VI-A.

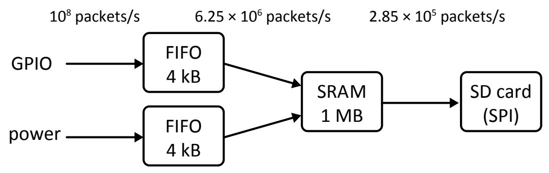

Figure 4. Data flow and memory structure of the data acquisition system. The SRAM serves as buffer for the SPI bus. The two data streams are merged when writing to the SRAM.

D. Configuration and Test Management

The TRACELAB board can be configured over a UART interface, which is connected to the USB port of the Gumstix using a USB-to-serial converter. The choice of a dedicated configuration interface (in addition to the SPI bus) facilitates the software implementation on the Gumstix, as data acquisition and configuration can be handled independently. To configure the FPGA, commands are provided to start or stop a test, to set a mask for target pins to be traced, and to control power profiling. During a test, commands can be sent to set or clear 3 actuation pins on the target. These commands are kept in a FIFO queue on the FPGA and processed at the specified time instant. In order to ensure proper actuation timing, an actuation command has to be sent at least 70 μs in advance.

E. Time Synchronization for Distributed 1-PPS

As described in Sec. IV-B, the internal system time on the FPGA is steered by applying an external 1-PPS signal. In TRACELAB, we leverage the fact that observers are placed within communication range of low-power wireless transceivers. Therefore we can employ a wireless mesh network built of such transceiver nodes to generate a synchronized 1-PPS signal on all observers. In TRACELAB, we use the CC1101 transceiver of the CC430 SoC for that purpose. A 1-PPS signal of a single GPS receiver serves as reference for the initiator node in the network. The remaining nodes synchronize to the initiator by means of a time synchronization protocol.[2]

Glossy [1] is a flooding architecture for wireless sensor networks that exploits constructive interference for fast network flooding and implicit time synchronization. On a TelosB node it achieves an average time synchronization error below one microsecond and is therefore suitable for our purpose. In Glossy, every node estimates the start time of a flood, based on timestamps made from several packets within the flood. This estimate serves as a *reference time*. Due to redundant use of links and retransmissions, Glossy achieves a high reliability.

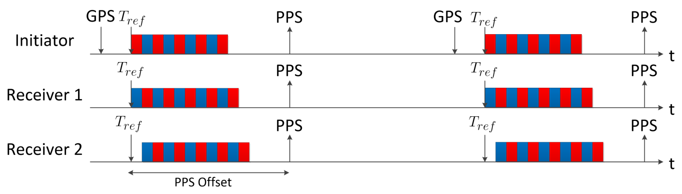

Figure 5. Generation of a synchronized 1-PPS signal. An external GPS pulse triggers the start of a flood at the initiator. The 1-PPS pulse is then emitted on all the nodes in the network based on the calculated reference time T_{ref}. This process is repeated every second.

Time synchronization in TRACELAB is illustrated in Fig. 5. The initiator starts a flood with every GPS pulse, *i.e.*, there is flood happening every second. We configure the GPS receiver to emit the pulse slightly *before* the start of a new second in order to align the node generated pulses *with* the start of a second.

Due to different influences like measurement uncertainties or different propagation paths, the calculated reference time is affected by jitter. To reduce this jitter, we apply a heuristic that exploits the fact that the local clock, running from a quartz oscillator, is relatively stable during shorter periods of time. The intuition is to combine every new reference time $T_{ref,i}$ with the measurement $\hat{T}_{ref,i-1}$ from the previous flood and weight them according to some smoothing factor α:

$$\hat{T}_{ref,i} = \alpha T_{ref,i} + (1-\alpha)(\hat{T}_{ref,i-1} + \bar{T}) \tag{1}$$

$\bar{T}$ is the average interval between the last recent M reference times.

[2]Potential in-band interference with target nodes can be avoided by blacklisting the frequency band/channel of the time synchronization protocol.

We evaluate the performance of the distributed synchronization pulse in Sec. VI-B and quantify the impact of our heuristic.

V. Implementation

The hardware implementation of a TraceLab board is shown in Fig. 6. It fits between a FlockLab board and a Gumstix and therefore extends an existing FlockLab observer in a modular fashion.

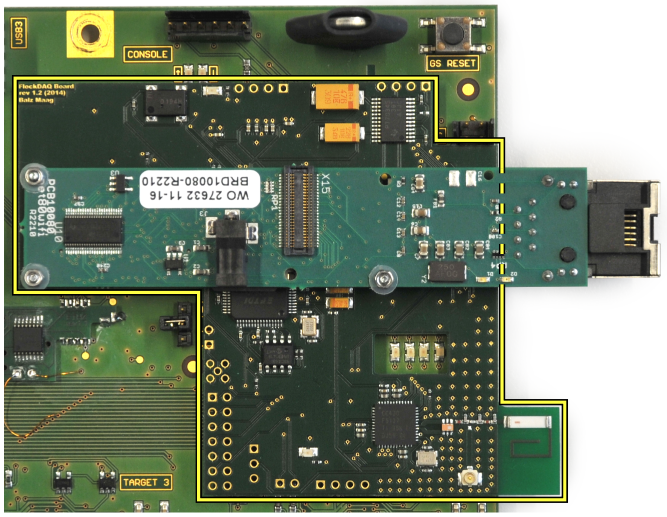

Figure 6. The TraceLab board fits between the FlockLab board and the Gumstix. Visible on the lower right part is the CC430 with a chip antenna. The FPGA and the SRAM chip are on the bottom layer.

We implement the data acquisition part on an FPGA of the Xilinx Spartan-6 series. The design fits into a Spartan-6 LX9, which is second smallest member of that family, featuring 120 user I/Os, 9152 logic cells and a price of about 15$ per piece. The network time synchronization protocol runs on a Texas Instrument CC430F5137 SoC, featuring 32 kB of program memory and 4 kB of RAM. The chip integrates a sub-1 GHz radio with configurable bit rate and radio modulation. A 26 MHz quartz oscillator provides the basis of a stable 13 MHz system and timer clock. A 4-port USB-to-serial converter connects the debug and programming ports of the FPGA and the CC430 with the Gumstix computer.

On the CC430, we run Glossy on top of the Contiki OS [13]. As suggested by [1], we exploit the automatic RX/TX-transceiver mode switch of the CC430 for accurate timing of concurrent transmissions in Glossy. Packets are sent using 250 kbit/s GFSK radio modulation in the 868 MHz frequency band.

VI. Evaluation

In a first part of the evaluation, we focus on a single observer node of TraceLab. We measure peak and average throughput of the data acquisition system and assess the performance of the 1-PPS-tracking algorithm on the FPGA. Next, we quantify the time synchronization error of the distributed time pulse for a setup of 31 observers in an office environment, using GPS receivers as ground truth. By running experiments with and without our jitter reduction algorithm, we show the beneficial impact of the algorithm. Finally, we assess the overall timing accuracy of TraceLab by using the wirelessly distributed time pulse as input to the 1-PPS port of the FPGA.

A. Throughput

In this section, we quantify the throughput of the data acquisition system and compare it to the requirements given in Sec. III.

As described in Sec. IV-C, the internal data path on the FPGA consists of several stages with different bandwidths. Here, we characterize the two maximal event rates that lead to a saturation of the first and second stage shown in Fig. 4, that is the FIFO queues and the SRAM. We empirically measure the number of events that can be processed without loss at event rates of 10^8 events/s and 6.25×10^6 events/s respectively. For this purpose, we connect an event generator to the tracing inputs of the FPGA and let the pin levels change at a constant rate. To detect the first lost packet, we compare received packets at the Gumstix with the generated events.

The results of this experiment are summarized in Table II. The FIFO queue, which can store up to 1024 event packets, is saturated after 1070 events at a constant event rate of 10^8 events/s. While filling the FIFO, data packets are continuously removed and written to the SRAM. For the second stage, the SRAM, we generate events at a rate of 6.25×10^6 events/s. 270,000 data packets can be stored until the first packet is dropped. The SRAM is full within 40 ms. The maximal average data rate that the data acquisition system can handle is determined by the SPI bus, which is the slowest interface in the data path.

With a continuous throughput of 285,000 packets per second, TraceLab is able to trace programs with 1.48 % tracing instructions on all target platforms in FlockLab (see Table I). The peak processing throughput of 10^8 events/s is high enough to meet the requirements of all target platforms.

Table II
MEASURED THROUGHPUT BURST SIZES AND MAXIMUM CONTINUOUS RATE

Cycles between two events	**Max. event burst size**
1 (10 ns)	1070
16 (160 ns)	270000
350 (3.5 μs)	continuous

B. Timing

In this section, we first assess the performance of the clock control algorithm on the FPGA. Then, we proceed to the evaluation of the distributed 1-PPS signal in a network of 31 observers and finally, we quantify the overall system performance of TraceLab in terms of time synchronization error between observers. In the experiments, we use one or several u-blox LEA-6T GPS receivers that provide an accurate 1-PPS signal (RMS of 30 ns) [14], either as a reference signal for the root node, or as ground truth.

Timing on the Single Observer. As described in Sec. IV-B, the implemented clock control algorithm on the FPGA seeks to correct the offset between the internal and the external 1-PPS signal. The external signal is provided by a GPS receiver. To evaluate the performance of the algorithm, we measure the time difference between those two signals in system ticks (*i.e.*, 10 ns) on a single TRACELAB observer for a period of 5.5 h.

The cumulative distribution function over all measurements is shown in Fig. 7. In total, 19,713 offset measurements are made, with a 99th percentile of 40 ns, which corresponds to 4 FPGA clock ticks. Therefore, we conclude that our control algorithm keeps the system time on the FPGA within tight bounds if a proper external 1-PPS signal is applied.

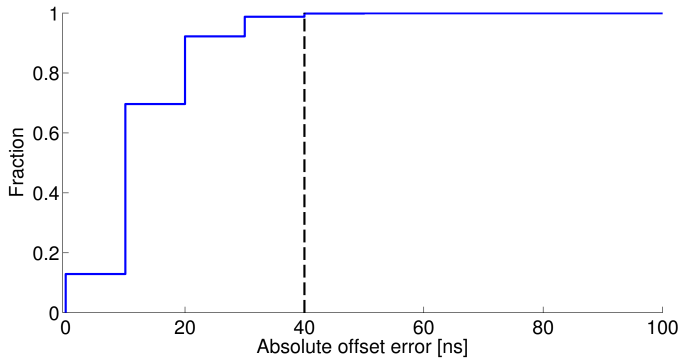

Figure 7. Cumulative distribution function of the absolute offset error between the internal and external 1-PPS signal. The clock tracking algorithm keeps the offset error within ±40 ns for 99 % of the time.

Network-Wide Time Synchronization. Next, we assess the accuracy of the distributed time pulse in a network setting of 31 nodes. This section focuses solely on the implementation on the CC430 SoC. The experiment is carried out on Olimex's commercially available MSP430-CCRF development board. In total, we distribute 31 nodes as shown on the floor plan in Fig. 8. 4 nodes are located outdoors while the remaining 27 are placed indoors in an office environment. We select a central node next to a window as initiator to keep hop distances short and to ensure good satellite signal for the reference GPS receiver. On the nodes, we run two different versions of Glossy: a baseline implementation *without* jitter reduction, and a version *with* jitter reduction, as described in Sec. IV-E. We set the smoothing factor α to 0.1.

To assess the synchronization error, we equip 5 nodes with additional GPS receivers. On these nodes, the GPS 1-PPS signal serves as ground truth. On every node, we locally measure for every Glossy flood the offset between the calculated reference time and the edge of the externally applied 1-PPS signal. We then compare all offsets relative to the initiator node to get the synchronization error. To ensure a broad coverage of environmental conditions, such as closed office doors, working people and temperature variations, we combine measurements originating from various daytimes and weekdays into a total measurement duration of 6 h. The results, summarized in Table III, show that we are able to keep the standard deviation of the synchronization error below 5 clock

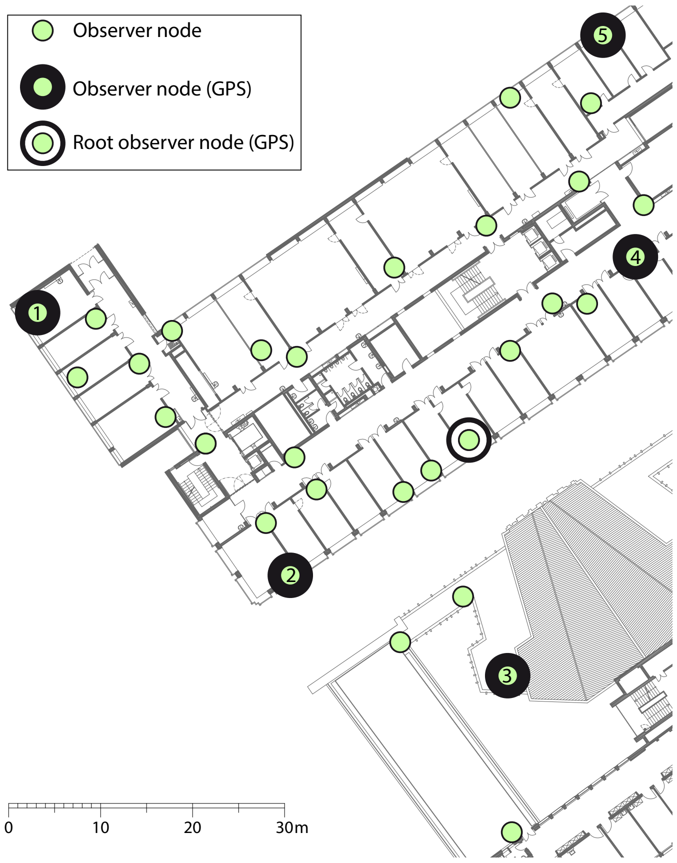

Figure 8. Layout of the network during evaluation of time synchronization.

ticks (385 ns) in the baseline implementation. Nodes that have a higher hop distance to the initiator exhibit a larger error. The outdoor node 3 has mostly a direct connection to the initiator and therefore the smallest error. The experimental data also shows a clear benefit of the jitter reduction heuristic. The maximal error as well as the standard deviation is considerably lower when jitter reduction is enabled. This effect is more prominent for nodes that are farther away from the initiator node. The error distributions for both implementations are exemplary illustrated for node 4 in Fig. 9.

Based on our results, we conclude that our wireless 1-PPS distribution infrastructure is well suited to synchronize observers in TRACELAB with sub-microsecond timing error.

Table III
STANDARD DEVIATION AND RANGE OF THE ERROR, MEASURED IN CLOCK TICKS (13 MHZ), FOR ALL NODES, WITHOUT AND WITH JITTER REDUCTION HEURISTIC.

Node	Glossy	Glossy with jitter reduction	Hop distance
1	4.05, [-25,82]	1.88, [-6,14]	4
2	2.15, [-11,17]	2.01, [-7,8]	2
3	1.94, [-7,7]	1.71, [-8,8]	1
4	3.25, [-23,23]	1.82, [-6,9]	2
5	3.62, [-22,32]	2.02, [-8,9]	4

Overall Timing Accuracy of TRACELAB. In the last experiment, we combine both the FPGA-design and the distributed

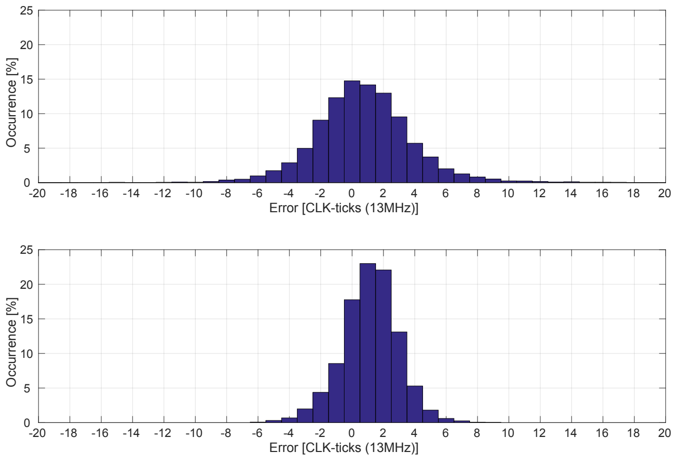

Figure 9. Distribution of synchronization error on node 4. The error distribution for the baseline implementation (top) is significantly broader than for the jitter reduced version (bottom).

time pulse to accurately trace GPIO events. The observers are placed in the same layout as in Sec. VI-B. Again, we equip the root observer and 5 other observers with a GPS receiver. The CC430 SoC on the root observer uses the 1-PPS signal of the GPS as reference and distributes the pulse to all other observers using Glossy with jitter reduction. On all the GPS-equipped observers, we connect the 1-PPS signal of the GPS to one of the tracing inputs of the TRACELAB board. Then, we configure the data acquisition system to trace this pin for a duration of 1 h. The data acquisition on the FPGA annotates every state change on the input pin using a globally synchronized timestamp. The difference between timestamps of different observers directly reflects the synchronization error. To evaluate the synchronization accuracy, we calculate for every pulse the maximal absolute error relative to the root observer. Fig. 10 shows a histogram of the error. 99.9 % of the errors are smaller or equal to 1 μs; the maximal error is 1.5 μs.

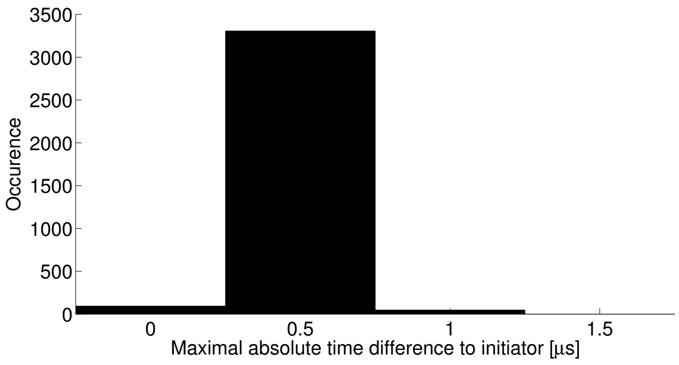

Figure 10. Absolute time synchronization error of TRACELAB. The histogram shows the maximal error of 5 out of 31 observers, relative to the root observer.

The reported error reflects the synchronization error between nodes on the edge of the network and the central root node. As the synchronization error in Glossy depends on hop distances [1], we expect synchronization between neighboring nodes to be even better. Overall, TRACELAB provides distributed event tracing services with tightly synchronized time information.

VII. CONCLUSIONS

We have presented TRACELAB, a testbed that allows to trace time sensitive system behavior of low-power wireless embedded systems in a fine-grained manner. By extending the existing FlockLab architecture with an accelerated data acquisition system based on an FPGA chip, TRACELAB is able to capture state changes at the maximal rate emitted by any of the currently attached target platforms. TRACELAB synchronizes traced data using a highly accurate wireless time synchronization protocol, thus enabling accurate monitoring of network interaction between all target nodes of the testbed, down to microsecond granularity.

ACKNOWLEDGMENTS

The work presented in this paper was scientifically evaluated by the SNSF, and financed by the Swiss Confederation and by nano-tera.ch.

REFERENCES

[1] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh, “Efficient network flooding and time synchronization with Glossy,” in *Proceedings of the 10th international Conference on Information Processing in Sensor Networks (IPSN)*, 2011.

[2] V. Sundaram, P. Eugster, and X. Zhang, “Efficient diagnostic tracing for wireless sensor networks,” in *Proceedings of the 8th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2010.

[3] P. Sommer and B. Kusy, “Minerva: Distributed tracing and debugging in wireless sensor networks,” in *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2013.

[4] M. Tancreti, M. S. Hossain, S. Bagchi, and V. Raghunathan, “Aveksha: A hardware-software approach for non-intrusive tracing and profiling of wireless embedded systems,” in *Proceedings of the 9th ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2011.

[5] 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 *Proceedings of the 12th international Conference on Information Processing in Sensor Networks (IPSN)*, 2013.

[6] D. Mills, J. Martin, J. Burbank, and W. Kasch, “Network time protocol version 4: Protocol and algorithms specification,” RFC 5905, 2010.

[7] “IEEE standard for a precision clock synchronization protocol for networked measurement and control systems,” *IEEE Std 1588-2008 (Revision of IEEE Std 1588-2002)*, pp. c1–269, July 2008.

[8] M. Tancreti, V. Sundaram, S. Bagchi, and P. Eugster, “Tardis: Software-only system-level record and replay in wireless sensor networks,” in *Proceedings of the 14th international Conference on Information Processing in Sensor Networks (IPSN)*, 2015.

[9] N. Penneman, L. Perneel, M. Timmerman, and B. De Sutter, “An FPGA-based real-time event sampler,” in *Reconfigurable Computing: Architectures, Tools and Applications*, 2010, vol. 5992, pp. 364–371.

[10] L. Gasparini, O. Zadedyurina, G. Fontana, D. Macii, A. Boni, and Y. Ofek, “A digital circuit for jitter reduction of GPS-disciplined 1-pps synchronization signals,” in *Advanced Methods for Uncertainty Estimation in Measurement, 2007 IEEE International Workshop on*, 2007.

[11] M. Maroti, B. Kusy, G. Simon, and A. Ledeczi, “The flooding time synchronization protocol,” in *Proceedings of the 2nd ACM Conference on Embedded Networked Sensor Systems (SenSys)*, 2004.

[12] T. Schmid, P. Dutta, and M. B. Srivastava, “High-resolution, low-power time synchronization an oxymoron no more,” in *Proceedings of the 9th international Conference on Information Processing in Sensor Networks (IPSN)*, 2010.

[13] “The Contiki operating system,” http://www.contiki-os.org/.

[14] *LEA-6, u-blox 6 GPS Modules Data Sheet*, u-blox.