Bluetooth Low Energy with Software-Defined Radio

Proof-of-Concept and Performance Analysis

Casparsen, Andreas; Christensen, Jonas Ingerslev; Antoniou, Panagiotis; Remy, Maxime Jérôme; Leyva-Mayorga, Israel; Madueño, Germán Corrales; Nielsen, Jimmy Jessen

Published in:
2023 IEEE 20th Consumer Communications and Networking Conference (CCNC)

DOI (link to publication from Publisher):
10.1109/CCNC51644.2023.10060476

Publication date:
2023

Document Version
Accepted author manuscript, peer reviewed version

Bluetooth Low Energy with Software-Defined Radio: Proof-of-Concept and Performance Analysis

Andreas Casparsen*, Jonas Ingerslev Christensen†, Panagiotis Antoniou,
Maxime Jérôme Remy‡, Israel Leyva-Mayorga*, Germán Corrales Madueño*‡, and Jimmy Jessen Nielsen*
*Department of Electronic Systems, Aalborg University, Aalborg, Denmark
†Samsung Denmark Research Center Aps, Aalborg, Denmark
‡Keysight Technologies Denmark Aps, Aalborg, Denmark
Emails:{aca, ilm, jjn}@es.aau.dk, ji.christens@samsung.com, {maxime.remy, german.madueno}@keysight.com,

***Abstract*—Software-Defined Radios (SDRs) enable more flexible connectivity solutions than traditional systems, but still face several challenges hindering their widespread adoption. General-Purpose Processor (GPP) based SDRs have generally been too slow for low-latency protocols. Meanwhile Field Programmable Gate Array (FPGA)-based SDR setups suffer from high prices and a steep learning curve for developers. This paper investigates the feasibility of implementing Bluetooth Low Energy (BLE) in a GPP based Peripheral Component Interconnect Express (PCIe) connected SDR. In particular, we focus on adhering to the timing requirements of BLE in a practical SDR implementation. For this, we propose a multi-threaded implementation based on a subset of the open-source BLE library BTLE. Using a signal generator and oscilloscope, we show that the SDR is able to achieve a response time down to 105 µs and can accurately respond in the required 150 ± 2 µs Inter Frame Space (IFS) time window. Furthermore, we also validate that channel hopping is supported by the SDR-based platform. To the best of our knowledge, this is the first SDR implementation able to meet the IFS requirements of BLE, hereby leading the way for more complete fully software based BLE protocol stacks.**

***Keywords*—Bluetooth Low Energy (BLE), Proof of Concept (PoC), Software-Defined Radio (SDR), Timing.**

I. Introduction

Reconfigurability is a key trait of softwarization, or the umbrella term for software defined principles, Software Defined everything (SDx) [1]. This term encompasses the paradigm of software-defined functions and virtualization for dynamic operations and cost savings. It allows to easily adapt, scale, and update the network infrastructure by updating the software, as opposed to the common cost-inefficient practice of replacing the hardware components across the network elements.

Software-Defined Radios (SDRs) enables the same generic piece of hardware to be used for different wireless technologies, by simply changing the software controlling it. Such reconfigurability is key to achieving true "always connectedness" in Internet of Things (IoT)-systems, where it should be possible to connect any IoT device to the internet, regardless of the available IoT-technology providing coverage. An enabler of this is the concept of a general IoT gateway [2], whose purpose is to provide connectivity and enable data collection for IoT devices, potentially using different communication protocols. An example of such implementation is an Field Programmable Gate Array (FPGA)-based IoT gateway supporting four different protocols [3]. Also, an architecture for software defined functionalities in IoT has been proposed [4], emphasizing added values such as openness, and reduction of operational costs. The very nature of SDRs fits very well with the IoT gateway architecture given that the support for multiple protocols depends only on the software implementation.

The work by [5] describes different SDR platforms, the hardware necessary for different design principles, and their respective trade-offs. Four primary designs are considered, namely, General-Purpose Processor (GPP), Digital Signal Processor, FPGA and hybrid. The last three involve hardware suited for signal processing. GPP is considered in two ways: 1) a pure GPP with just a Central Processing Unit (CPU), or 2) a GPP with a CPU and, e.g., a Graphics Processing Unit (GPU) for signal processing. A particular challenge in pure GPP-based platforms is to achieve the timing requirements of typical IoT protocols [6]. In particular, it was observed in [6] that the typical SDR-to-host latency for diverse platforms is in the order of milliseconds, which is several magnitudes higher than required by protocols such as WiFi and Bluetooth Low Energy (BLE). Low latency protocols like BLE, with an Inter Frame Space (IFS) of 150 ± 2 µs, can generally only be realized by using additional hardware, for example, an FPGA. The IFS measured as the time between the end of a frame and the beginning of the succeeding frame. While, successful implementations of a GPP and GPU-based platform for tracking and decoding traffic across the entire BLE spectrum have been achieved [7], the protocol interactions have not been considered.

This work is based on a Proof of Concept (PoC) for a BLE Master using only an off-the-shelf CPU for the processing tasks. The main contributions of this work are:

- Multi-threaded software design that allows to adhere to BLE's strict lower layer requirements, including the IFS.
- Experimental verification and performance evaluation of the PoC to support a BLE implementation.

This work was fully conducted as part of the Master Thesis of Andreas Casparsen, Jonas Ingerslev Christensen, and Panagiotis Antoniou at Aalborg University.

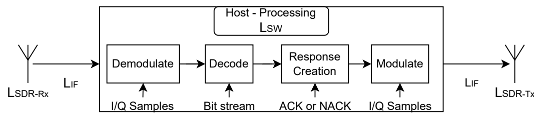


Figure 1: Model of software components and hardware components and latencies considered in an SDR implementation.

To the best of the authors' knowledge, this is the first instance of a successful GPP-based SDR setup only using the CPU for signal processing to enable low layer communication for BLE. The rest of the paper is organized as follows. Section II describes our SDR latency model. Section III describes the design and implementation of the software components Section IV shows hardware and full system performance. Section V concludes the paper.

II. Latency Requirements

The primary requirement of the developed PoC is that the response must be transmitted within the IFS interval $[148, 152]\,\mu$s. In an SDR implementation, both software and hardware processes introduce latency to the communication, which is non-deterministic and, hence, can be modeled as a random variable. For an acknowledgment to be made, first the analog data must be received and converted to digital form, with latency $L_{\text{SDR-Rx}}$, and passed through the SDR-to-host interface with latency L_{IF}. These processes are performed in hardware and the result is the input to the CPU. The samples are then processed by the CPU using a software implementation with overall latency L_{SW} to generate a response. The response is then given to the SDR through the SDR-to-host interface. Then, the data is converted back from digital to analog in hardware, which takes $L_{\text{SDR-Tx}}$. Data can be transmitted immediately afterwards. The complete process is illustrated in Figure 1, where all components within the box are implemented in software and those outside are implemented on hardware. Thus, as described in [6], the overall latency is given by the sum of random variables (RVs) $L_{\text{HW}} + L_{\text{SW}}$, where the latency at the hardware components is

$$L_{\text{HW}} = L_{\text{SDR-Rx}} + 2L_{\text{IF}} + L_{\text{SDR-Tx}}. \tag{1}$$

The necessary criterion to fulfil is that all steps in hardware and software, are completed within the IFS. Therefore, we define the reliability of an SDR implementation as

$$r_{\text{SDR}} = \Pr\left[148 \leq L_{\text{HW}} + L_{\text{SW}} \leq 152\right]. \tag{2}$$

That is, it is necessary to ensure that neither bounds of the IFS are violated. As shown in Fig. 1, the latency of the software components L_{SW} is constituted by the time the CPU takes to demodulate, decode, create and modulate the response:

$$L_{\text{SW}} = L_{\text{demodulate}} + L_{\text{decode}} + L_{\text{response}} + L_{\text{modulate}}. \tag{3}$$

The multi-threaded software implementation included in the PoC achieves the demodulation and decoding in parallel so that the symbols are processed one by one and the overall latency is reduced. Naturally, characterizing $T_{\text{HW}} + T_{\text{SW}}$ analytically is extremely complicated and goes beyond the scope of this paper. Nevertheless, we characterize the empirical distribution for the specific PoC in Section IV and estimate its reliability after fitting a well-known distribution to the empirical data.

III. System Design and implementation

The software implementation is based on a third-party library BTLE [8]. The original version contained individual RX and TX modules, which can be found on the Github [8]. A sniffer for BLE and a module to create, modulate, and transmit BLE frames. Both were meant for an implementation with HackRF or BladeRF. Modifications were made to the original BLTE library to reduce the processing time to meet the IFS requirement. Specifically, with our configuration for the SDR the sniffer could not operate in real time, and the other module took 170ms to create I/Q samples. Our BLE implementation is divided into modules that can run in parallel and, hence, minimize the idle periods on the CPU. The modules and their internal interaction are specified in the following.

- **Rx_Thread**: Continuously read samples from the SDR's antenna and store them in a buffer for Mod_Thread
- **Mod_Thread**: Receive samples from Rx_Thread and start detecting access address. If found, demodulate the header and extract packet length, and immediately start ACK & NACK response. Depending on Cyclic Redundancy Code (CRC) give response to TX_Thread.
- **Tx_Thread**: Receive samples and response transmission time from Rx_Thread. When transmission time is, forward the samples to the antenna and transmit the samples.

A timeline perspective is shown in Figure 2, which illustrates how parallelization helps reduce the response time. The average execution time of each module is listed above each function. These were measured individually in microseconds, with little to no observed variation between measurements. Two things to note is that, firstly the Payload parsing and CRC check start only when the entire packet is received. Secondly, demodulation of received samples occur before any further decoding of the frame. The execution time is measured when all bits associated with that part of the frame are demodulated. Demodulation gains the most from parallelization as samples are received parallel to the construction of the received frame. Samples must always be received to not lose data. Samples are received every 3 µs, which resulted in the highest stable rate for the platform, and the packet is built discretely over time. At the end of the frame 8 µs are spent for CRC to pick ACK or NACK. Scheduling the response and the write latency are 106 µs combined. The total latency of 117 µs remains well below the IFS, leaving a comfortable headroom that allows for dealing with variations in modulation time or variation in software execution time. We note that demodulation and modulation steps are executed sequentially in the Mod_Thread.

IV. Results

The platform utilizes an off the shelf PC with i9-9900k processor, 32GB DDR4-2666 RAM and a heatsink for cooling. Initially, the platform was tested with respect to the time it takes to receive and respond to a signal, without any signal

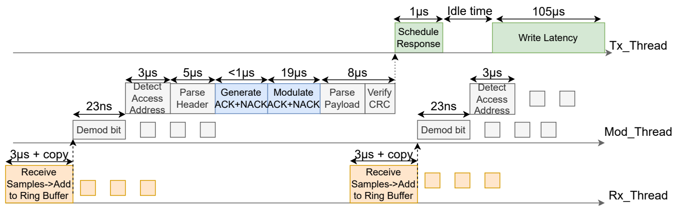


Figure 2: Threaded perspective of function execution with their average execution time. New samples are retrieved every 3 µs.

processing or processing of a response. This result indicates the limitations of the hardware platform itself. After this, capabilities of the platform and protocol implementation to stay within the IFS were analyzed. To characterize the performance of the SDR a Signal Generator and Oscilloscope were used. In both tests, the Signal generator initially transmits a frame. In the first test, the SDR responds immediately to gauge the hardware latency. In the second test, the SDR will aim to respond within the IFS to measure the accuracy. Components used for the tests included an MSOX4154 oscilloscope, an N5182A MXG signal generator.

Traditionally for GPP-based SDRs is the L_{HW} limiting factor. By retrieving samples more often it can be reduced, and the SDR has one such feature called DMA_buffer_size for this purpose. By setting it small, we retrieve samples often and found that 3 µs being optimal for stability and responsiveness. If too small the program would crash. Different values for the sample rate were also tested. It was found that at 40 MHz the performance was stable and highly responsive. Evaluation of the L_{HW} is based on sending a frame from the signal generator. A response is made once an increased voltage is observed at the host. This involves minimal host processing, and is practically only the hardware. Figure 3, shows the achieved response time of 105 µs. The first increase on the yellow signal (SDR) is leakage from the signal generator. 105 µs after the green signal(generator) is high the yellow signal goes high as the response is received at the oscilloscope. While the response-time is found to be around 105 μs, further investigation found that the latency of samples is not always 3 µs, as expected from the DMA_buffer_size, effectively making it less responsive.

Latency of new samples, returned by the read function, ranged from 3 to 40 µs. We noted that 99.14% of all these occur at $3\mu s$ and $4\mu s$, hence late samples are rare. The effective responsiveness is therefore between 105 and 142 µs when accounting for stuck samples, waiting to be retrieved and processed. While late samples influences the performance for the PoC, it is primarily an issue at the end of the frame. It should be emphasized that the worst case of 40 µs happens very rarely – it was only observed 2 times over 100 million samples. A thing to note in this context is that performance evaluation only occurs on executions of the SDR where it performs *correctly*. When started, the SDR may perform sub-par, with read operations taking more than 100 μs. This is, however, fast to measure and find out within the first minute. If it does not happen in this period, it does not happen afterward. It was found out in run-time, and the SDR rebooted

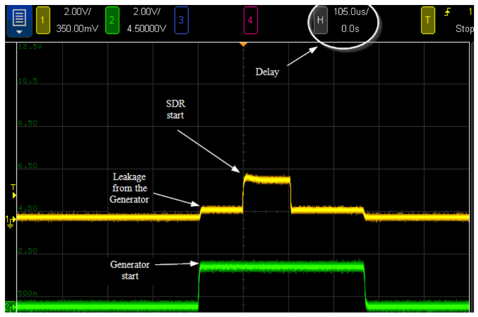
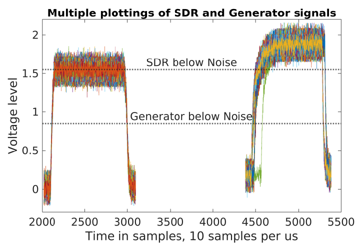


Figure 3: Fastest response-time measure can be seen at the H.

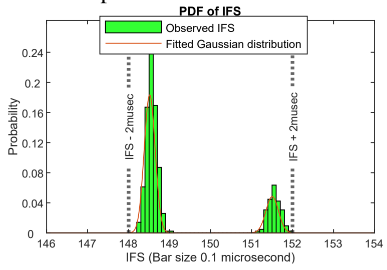
automatically until a *correct* instance is found. For that reason, only the *correct* instances are evaluated.

The response-time performance of the full implementation is analyzed by using an oscilloscope to measure voltage levels. In Figure 4a all signals from the signal generator and SDR have been plotted on top of each other by applying the modulus function on the data with the generator periodicity interval of 460 µs. While the rising and falling edges of the voltage profile of the signal generator are very similar for all instances, the SDR response signal shows temporal variation at the beginning and end of the transmission. On the falling edge, two different time offsets separated by 3 µs can be distinguished. The same is visible at the rising edge, however visually less apparent due to the smoother rise. Finally, we noticed an outlier (shown in green) that is too short to be a correct frame can also be seen. We believe this is an example of one of the last samples being late, thereby causing delayed start of transmission. However, since the end of transmission is specified as an absolute timestamp in the code, the transmission ends at the designated time resulting in an incomplete frame. From the measurements, we cannot immediately read the response-time, which is the time between the end of the generator signal until the SDR response starts.

First, to find the end time of the generator signal, we use the knowledge that the transmitted signal from the generator is 88 µs long. Centering this interval between the flanks of the generator signal, we can determine the end time. As the SDR response has a smooth rising edge, we consider the end of the signal to be when the falling edge goes below the level of variation around the mean in the high part of the signal. From this point, we subtract the duration of the response signal, 80 µs, to get the start timestamp of the response signal, from which the response time is calculated. A PDF of the resulting response time measurements can be seen in figure 4b, where 424 of the frames were correctly placed within the IFS. The outlier from previously is not shown as it was incorrect, it was too small and considered an automatic failure. This leads to the performance of the implementation upholding the IFS in 424 out of 425 instances. All 424 correct frames were within the IFS $\pm$ 2 µs, showing that the SDR implementation fulfill the IFS requirement. Further, we notice that the distribution resembles that of a mixture of two Gaussian random variables: $X = c_1 X_1 + c_1 X_1$ where $X_i \sim \mathcal{N}(\mu_i, \sigma_i^2)$ and c_i are mixing coefficients, $i \in \{1, 2\}$. The reason for this characteristic is



(a) Observed signals for the 425 SDR frames, plotted on top of each other. Each color corresponds to a different frame.



(b) Empirical distribution of the response time obtained from PoC measurements for the correct frames and fitted Gaussian distribution.

Figure 4: Measurements of signals and response time.

Table I: Fitted parameters of Gaussian mixture distribution and their 95% confidence interval.

i	c_i	μ_i	σ_i
1	0.77	$148.49 \leq \mu \leq 148.53$	$0.124 \leq \sigma \leq 0.145$
2	0.23	$151.47 \leq \mu \leq 151.54$	$0.134 \leq \sigma \leq 0.179$

uncertain, but could be related to oscilloscope resolution. The Gaussian distributions have been fitted individually, resulting in the parameters listed in Table I.

We notice that the fitted distributions are offset by 3 µs, corresponding to the observed difference in the falling edge of the SDR response, seen in figure 4a. A chi-square goodness of fit test was performed on each of the Gaussian components. The latter is a statistical hypothesis test used to evaluate how likely observed data are to have been sampled from a hypothesized distribution. A p-value, with $p \in [0, 1]$, is used as a threshold to determine if a null hypothesis can be rejected. For our definition, the null hypothesis is that the observed data are sampled from the fitted distribution. Typically, the null hypothesis can only be rejected if $p \leq 0.05$ and the resulting p-values for our goodness of fit test are 0.77 and 0.84 for the respective distributions. With these p-values, it is safe to conclude that the values are sampled from the fitted Gaussian distributions, and use this assumption for further evaluations. Following, the probability of upholding the IFS requirement, given by $a = 148$ and $b = 152$, can be calculated as:

$$\begin{aligned}\Pr[a < X < b] &= c_1 \Pr[a < X_1 < b] + c_2 \Pr[a < X_2 < b] \\ &= \frac{c_1}{\sigma_1\sqrt{2\pi}} \int_a^b \exp\left(-\frac{(t-\mu_1)^2}{2\sigma_1^2}\right) dt \\ &\quad + \frac{c_2}{\sigma_2\sqrt{2\pi}} \int_a^b \exp\left(-\frac{(t-\mu_2)^2}{2\sigma_2^2}\right) dt \\ &= 0.9998. \end{aligned} \tag{4}$$

The probability is evaluated to give rise to a more accurate performance evaluation of the full implementation, when samples are not delayed as also previously seen. This is especially required due to the small sample size of 424, from where our estimated distribution, with a good fit, suggests an error probability of $2 \cdot 10^{-4}$. This error type is caused by the inaccuracies in when the ACK frame starts.

Frequency hopping was also tested with N9020B MXA signal analyzer by shifting the I/Q samples from the center frequency to the new channel. We found that we can shift these before transmission, by pre-computing the shift in samples. With the current configuration we support half the BLE spectrum with 40 MHz sampling, as the SDR's AD9361 chip samples orthogonally meaning effectively at 80 MHz. We could likely fully support the entirety with 2 individual SDR.

V. Conclusion

In this paper, we presented the design of our PoC for a BLE Master using a GPP-based Peripheral Component Interconnect Express (PCIe) SDR. The PoC is based on a multi-threaded partial implementation of the BLE controller, which enables low granularity of sampling and fast demodulation. The results show a $2 \cdot 10^{-4}$ error rate for upholding the IFS while supporting the channel hopping functionality of BLE. These results highlight the advancements made possible with new PCIe-SDRs and serve as a stepping stone towards implementing the full BLE stack in SDRs, which is not yet supported. Future directions include combining multiple SDRs to cover the entire spectrum of the BLE and systematic testing to compare the SDR implementations against commercial BLE products, and ensure full protocol stack support. Besides, the promising results indicate that our software design can be adapted to other protocols not currently supported in SDR such as LTE, LoRa, 802.11ah. Ultimately, SDR solutions might lead to a fully reconfigurable IoT gateway, where support for new protocols is added through software updates.

References

[1] Deloitte. Software-definedeverything. [Online]. Available: https://www2.deloitte.com/content/dam/Deloitte/us/Documents/financial-services/us-fsi-software-defined-everything.pdf

[2] H. Chen, X. Jia, and H. Li, "A brief introduction to IoT gateway," in *Proc. IET International Conference on Communication Technology and Application (ICCTA 2011)*, 2011, pp. 610–613.

[3] C. Gavrilă, C.-Z. Kertesz, M. Alexandru, and V. Popescu, "Reconfigurable IoT gateway based on a SDR platform," in *Proc. International Conference on Communications (COMM)*, 2018, pp. 345–348.

[4] T. Ahmed, A. Alleg, and N. Marie-Magdelaine, "An architecture framework for virtualization of iot network," in *Proc. IEEE Conference on Network Softwarization (NetSoft)*, 2019, pp. 183–187.

[5] R. Akeela and B. Dezfouli, "Software-defined radios: Architecture, state-of-the-art, and challenges," *Computer Communications*, vol. 128, pp. 106–125, 2018.

[6] D. M. Molla, H. Badis, L. George, and M. Berbineau, "Software defined radio platforms for wireless technologies," *IEEE Access*, vol. 10, pp. 26 203–26 229, 2022.

[7] M. Cominelli, P. Patras, and F. Gringoli, "One GPU to snoop them all: a full-band Bluetooth low energy sniffer," in *Proc. Mediterranean Communication and Computer Networking Conference (MedComNet)*, 2020.

[8] J. Xianjun. BTLE. [Online]. Available: https://github.com/JiaoXianjun/BTLE