Real-time Systems / Embedded Networked Systems

WS ’24/’25

Wireless Logic Analyzer  
*(project name)*

author: Tsvetoslav Valentinov Mavrodiev, 03775903

course: IN2060

1. Project description

The idea is to create a simple logic analyzer on an Arduino Nano 33 BLE microcontroller utilizing all 14 digital IO pins (with configurable external interrupts).

The most useful feature would be configuration of sampling rate, setting up triggers and data acquisition over Bluetooth Low Energy connection. This way one could effortlessly connect to the LA using any device supporting BLE (e.g., a laptop or even a smartphone).

An additional feature may be connecting to multiple wireless logic analyzers using the same device which will increase the available inputs.

1. Hardware and software setup
   1. Hardware

For this project I chose to use a pair of Arduino Nano 33 BLE rev 2 microcontrollers since they are designed for wearable devices – they are small, energy efficient, and support wireless communication through Bluetooth Low Energy. Hence, they are suitable for the purposes of this project, namely debugging complex and physically large communication systems, controlling and data acquisition of industrial equipment and many others.

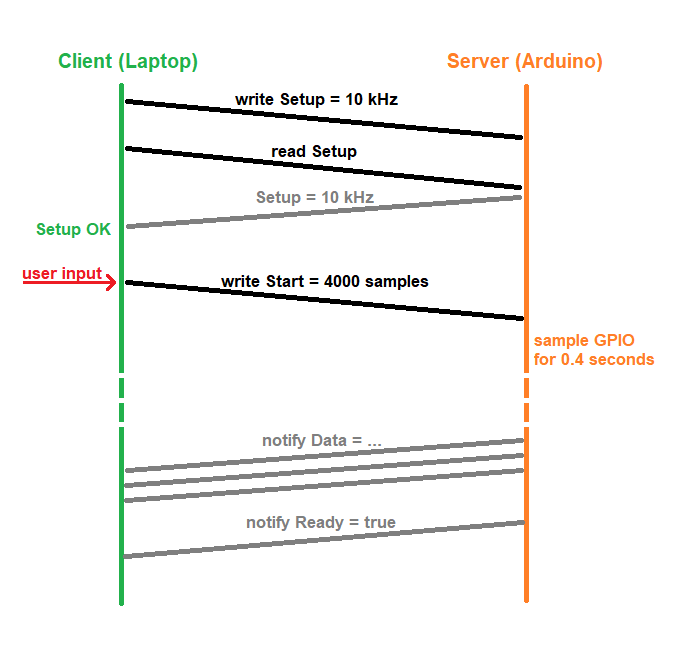
The development boards integrate the nRF52840 Processor of Nordic Semiconductor running at frequency of 64 MHz. The CPU also has 5 configurable 16 MHz hardware timers, which will allow us to sample the GPIOs at up to 1 MHz frequency without timing collisions. Furthermore, the chip has 256 KB of SRAM and 1 MB of flash memory.

* 1. Software

For the firmware development I went with the official Arduino IDE v2.3.4, since it is simple to use, as it utilizes the Arduino bootloader that comes preinstalled on the boards and doesn’t require additional programmer hardware to flash the firmware. Also, the Arduino framework is built on Mbed RTOS and the official nRF toolchain, thus I still have direct access to these tools while developing our software.

Additionally, I need to implement a minimal and very simple client interface, which will communicate with the Arduino boards to configure them and to extract the collected data. I’m planning to develop this application in Python using the “Bleak” library, which provides BLE GATT client interface, and “matplotlib” and “tkinter“ for visualizing the data in a GUI.

1. Implementation
   1. Communication structure

The communication structure is based on the standard Bluetooth Low Energy General Attribute Profile (GATT), which is the layer that defines how application data is transferred. The protocol operates between a “server” and a “client”, which in my case are respectively the Arduino board and my laptop with Bluetooth enabled. Initially, the server advertises its characteristics grouped in services. These characteristics are the endpoints of the communication, and the client is able to read, write and receive notifications from them after connecting to the server device.

Here are the available characteristics on the microcontrollers and what they do:

* SETUP (int, UUID = "5a61383f-e7e0-4e92-bc52-4ff803be0f1c")
  + When written: configures the hardware timer of the Arduino with the specified frequency, afterwards the sampling can begin.
* START (int, UUID = "3642a423-afd4-4ab1-95b2-5f7bff961610")
  + When written: initiates a sampling cycle with the provided number of samples, dividing this number by the sampling rate, the sampling period (in seconds) can be calculated.
* DATA (bytes, 20 B, UUID = "ea91a314-ab37-4f3b-8a58-d3660b5ab81a")
  + This is a read-only characteristic with notifications enabled, this means, once subscribed to it, the client can directly receive stream of data from the server without explicitly requesting to read the characteristic. It is used to send back the data, after the sampling completes.
* READY (bool, UUID = "d80ca4bd-7c02-4728-bf8a-3e6c10c4b378")
  + This is also a notifiable characteristic, which is used by the server to indicate that all the data from the sampling buffer has been sent successfully.
  1. Firmware

Key points in the implementation of the firmware for the boards were the Bluetooth LE communication and the sampling of the GPIO pins.

The former was quite straight forward to accomplish using the “ArduinoBLE” library and following the documentation and the provided examples. The UUIDs of the service and its characteristics I chose to be version 4 (i.e., random identificators) to minimize collisions with already existing services/characteristics and are generated using <https://www.uuidgenerator.net/>. Also something important, that should be noted is that according to the Bluetooth Core Specification the maximum length of a notifiable characteristic is only 20 bytes, which may slow down the stream communication a bit, but also makes it more reliable.

Sadly, neither the Arduino library, nor the MbedOS API provided an easy way of directly using the hardware timers on the CPU, but only quite inaccurate variants, that are affected by the task prioritization of the RTOS and cannot achieve a precise timing in high frequency operation (“mbed::Ticker” class). Hence, it is better to directly configure the timer registers using the nRF stack. [<https://forum.arduino.cc/t/five-hardware-timers-example/905798>]

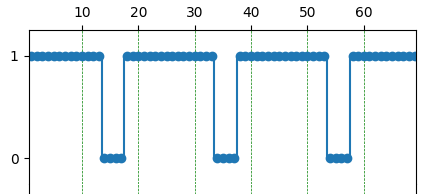
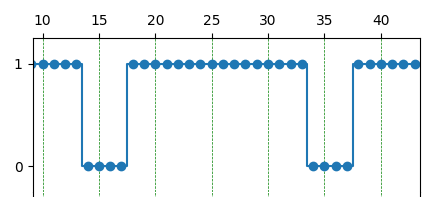
The challenge with the GPIO sampling was to come up with a way to efficiently obtain the state of each digital pin at the same time and in a timely manner, so that the hardware timer can run with higher frequency without locking up. An intuitive way of doing this is to call the function “digitalRead(pin)” provided by the Arduino library for each digital pin on the board. This, however, if done in an interrupt service routine (ISR), which is the hardware timer expiry callback, has the drawback of taking too much time to execute, leading to the problem mentioned above. The solution to it is to find a way to fetch the whole GPIO port state with a few instructions and then storing that in a buffer for further processing later. Luckily, the nRF stack helps us by providing the “nrf\_gpio\_port\_in\_read(port)” function, which allows us to read the state of every pin on the port that is configured as input. This approach is more efficient, because it reads the data stored subsequently in the memory with way less processor instructions.

* 1. Client software

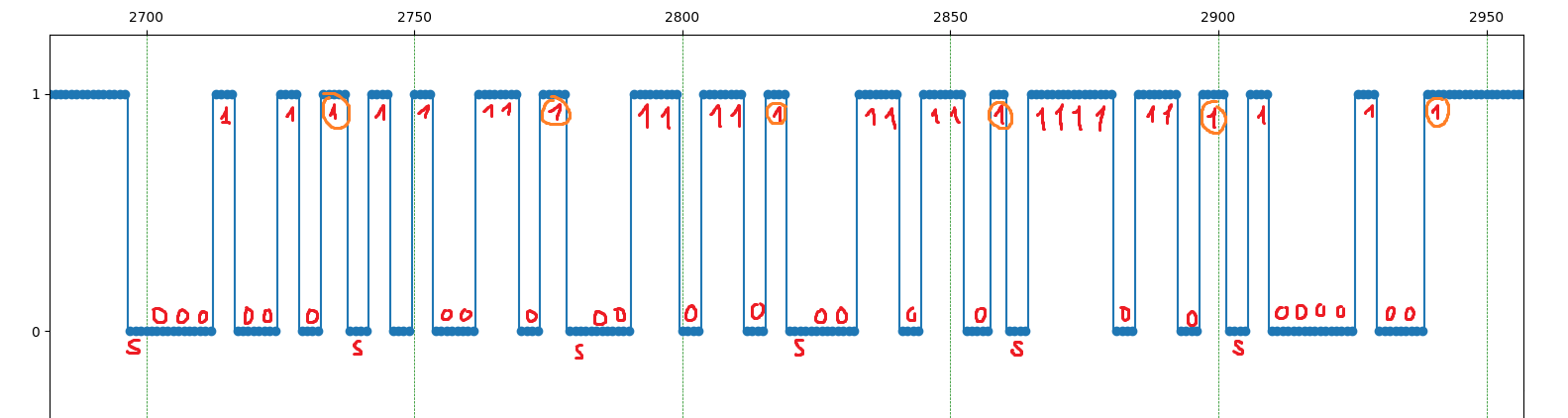
The client software runs on a laptop or a PC with enabled Bluetooth interface and support of the BLE specification. It is utilizing a “tkinter“ graphical user interface which automatically scans for and connects to a device with the designated name “BLELA” and afterwards enables the user to set up a sampling frequency and start a sampling cycle and then visualizes the data received on a plot navigated with scroll and left and right keyboard keys.

1. Results and analysis

I analyzed two test scenarios – sampling a PWM signal generated by the “analogWrite(200)” from another Arduino Nano 33 BLE board and examining an UART communication signal. Here are the results:



From the waveform (sampled at 10 ksps – thousand samples per second) can be calculated both the PWM frequency = 10 ksps / 20 samples period = 0.5 kHz = 500 Hz, which corresponds to the Arduino documentation, and the duty cycle = 16 / 20 = 80% (the actual one should be 200 / 255 = 78.4%). Higher sampling frequency would of course lead to greater accuracy.



Here is displayed the waveform of the UART message “Hello!” sampled at 40 ksps. The original signal is with baud rate of 9600 bps and follows the 8-N-1 convention (8 data bits, no parity bit and 1 stop bit). The start bits are marked with the letter ‘S’ and the stop bits are circled with orange. We can decode the letters using their ASCII codes and the fact that UART relays the data bits in order from the less significant to the most significant bit: 01001000(2) = 72 = ‘H’, 01100101(2) = 101 = ‘e’, 01101100(2) = 108 = ‘l’ and so on.

1. Conclusion

In conclusion the main objective of the project is achieved – development of a logic analyzer with the functionality of transmitting the collected data over Bluetooth Low Energy. There are still features that are not implemented but useful and present in commercial logic analyzers, for example: triggers, which are possible by configuring external interrupts on the GPIO pins, data acquisition functionality.

I should also mention a drawback of the current implementation that may be object to future research – the data transmission over the BLE connection. As of now the data rate is limited to around 1 KB/s and the data is not compressed nor optimized in any way prior to transmission. Furthermore, even if the user needs to sample only one or two channels, all the data for the other channels (pins D2 – D12 on the Arduino and overall 2 bytes per sample) is also relayed, allocating precious bandwidth.