# Introduction

The bachelor’s Thesis is realised in collaboration with HES-SO Valais/Wallis, Nordic Semiconductor and NTNU. All the project is developed at Nordic Semiconductor in Trondheim, Norway.

Nordic Semiconductor is specialized in the development of SoC and provide a large range of tool to develop application using Bluetooth. Therefore, The Bluetooth Low Energy, BLE, is the main element of this project.

The main idea of this project is to measure the behaviour of Zephyr RTOS with nRF5x series SoC from Nordic Semiconductor. Zephyr is a recent RTOS developed for IoT application and provides a Bluetooth API.

Generally, the Nordic’s customers use the SoC with Bare Metal system and using libraries, SoftDevice and Software development kit, developed by Nordic Semiconductor. But, Nordic Semiconductor is interested by the possibility of using a RTOS with its products.

However, it is important for Nordic Semiconductor that the requirements to use Zephyr RTOS and the Bluetooth Low Energy with high traffic are respected and to compare the performances with a Bare Metal system using the SoftDevice.

It is not a question of which one is a best but to analyse if Zephyr RTOS is a good solution for tiny embedded system using Bluetooth Low Energy.

Before stating this project, I had no idea about the Bluetooth Low Energy and I never used the SoftDevice, the Software Development Kit and Zephyr RTOS. Hence, it was interesting to see which systems were the most easier to learn and to implement.

# Why Zephyr RTOS?

Here is the description from Zephyr Project GitHub Repository.

*“The Zephyr™* *Project is a scalable real-time operating system (RTOS) supporting multiple hardware architectures, optimized for resource constrained devices, and built with security in mind.*

*The Zephyr OS is based on a small-footprint kernel designed for use on resource-constrained systems: from simple embedded environmental sensors and LED wearables to sophisticated smart watches and IoT wireless gateways.*

*The Zephyr kernel supports multiple architectures, including ARM Cortex-M, Intel x86, ARC, NIOS II and RISC V, and a large number of* [*supported boards*](https://www.zephyrproject.org/doc/boards/boards.html)*.”*



Figure : Zephyr RTOS logo

## Philosophy

Zephyr is Apache 2.0 licensed and has a fully open development model. The terms and conditions of this license are mainly:

* all copies, modified or unmodified, are accompanied by a copy of the licence
* all modifications are clearly marked as being the work of the modifier
* all notices of copyright, trademark and patent rights are reproduced accurately in distributed copies
* the licensee does not use any trademarks that belong to the licensor

Hence, it is possible to anyone to access to the source code and to modify it as he wants regards to the license’s conditions.

In addition of the Apache license, Zephyr Project source is maintained on a public GitHub Repository. GitHub allow anyone to be a contributor of the project by notifying bugs or by proposing improvements.

To contribute to the project, a programmer pulls a request for a bug or improvement that will be analysed by one or several main developers of the project. If the request is approved, the code of the request is merge with the source code of the project.

## Benefits for Nordic Semiconductor

Nordic is currently member of the Zephyr RTOS. Therefore, it is already possible to use Zephyr RTOS with nRF5x series SoC. However, it is already possible to use different RTOS as FreeRTOS or Keil with Nordic’s produces. So, why is it interesting for Nordic Semiconductor?

At first, Zephyr RTOS provides all the tools to develop a IoT application with Bluetooth, Networking, I/O drivers API.

Secondly, Zephyr RTOS supports multiple hardware architectures and it allows to easily port an application on new nRF5x series release without changing the Nordic’s SoftDevices and SDK version.

Then, Zephyr RTOS is an Open Source Project. Hence, it is possible to adapt easily the RTOS and to add tools for a specific project and to contribute to the project.

The last reason concerns the memory. The nRF5x series allows to use complex RTOS that require more RAM and Flash memory as Zephyr. More, Zephyr is a fusion between the SDK and the SoftDevice and therefore saves more Flash and RAM memory.

|  |  |  |
| --- | --- | --- |
|  | SoftDevice | Zephyr RTOS |
| Flash | 136kBytes | 47kBytes |
| RAM | 8kBytes | 14kBytes |

Table : Comparison of memory requirement

Note: The value for the SoftDevice are the value indicated in the examples of the nRF5 SDK v13.0.0.

## My Contribution

As Zephyr is a recent project, some tools, useful for my project, were not already provided by Zephyr. Therefore, I contributed to the Zephyr project during my Bachelor’s Thesis. My contribution is:

* Fix a bug on the nRF5x SPI driver configuration

Because of a lack of time, I had no time to pull other requests during the project. But it is planned to pull requests for the element I added for my project:

* nRF5840 GPIO Port 1 driver
* nRF5840 SPI2 driver
* Possibility to use the SPI and I2C on GPIO Port 1

## Sources

<https://www.zephyrproject.org/>

<https://github.com/zephyrproject-rtos/zephyr>

<https://nexus.zephyrproject.org/content/sites/site/org.zephyrproject.zephyr/dev/api/api.html>

<https://devzone.nordicsemi.com/blogs/1059/nrf5x-support-within-the-zephyr-project-rtos/>

<https://www.apache.org/licenses/LICENSE-2.0>

<http://oss-watch.ac.uk/resources/apache2>

# Specifications

To measure the behaviour of Zephyr RTOS and SoftDevice, it is important to define the use cases. As the Bluetooth Low Energy is at the centre of this project, the uses cases are defined by the BLE:

* **Peripheral**, one or several devices send data to a central
* **Central**, receive the data from peripheral and maintain the connection time

To realise the Bachelor’s Thesis, some materials are provided:

* **Nordic nRF52840 Development Kit** to implement the BLE peripheral and central
* **Nordic Power Profiler Kit** to measure the power consumption

The performances of the peripheral are tested with one central and one peripheral connected. As the BLE peripheral is the system that acquire data from sensor, an extension board is developed with different components:

* **A/D converter**
* **Accelerometer**
* **Interrupt generator**

Those devices allow to test the processor in real circumstances and to generate interruptions to stress the system. The frequency of the interruptions is incremented until the system crash. It is important that when system crash, it is still able to maintain the Bluetooth connection.

The peripheral must be able to maintain the connection with the central when the throughputs of the sensors and the BLE connection requirements are fast.

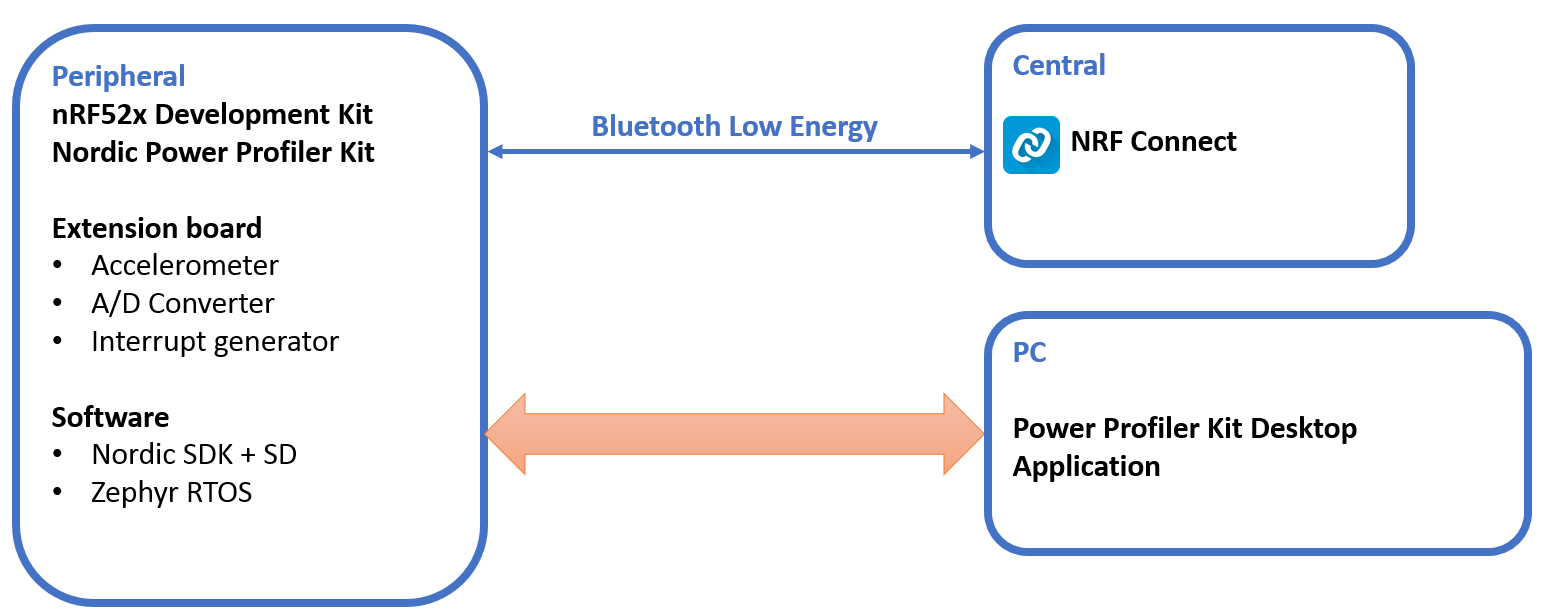


Figure : Behaviour measurement on BLE peripheral Schema Block

The performances of the peripheral are tested with one central and several peripherals connected. The central must be able to maintain the connection with all the peripherals when the BLE connection requirements are fast.

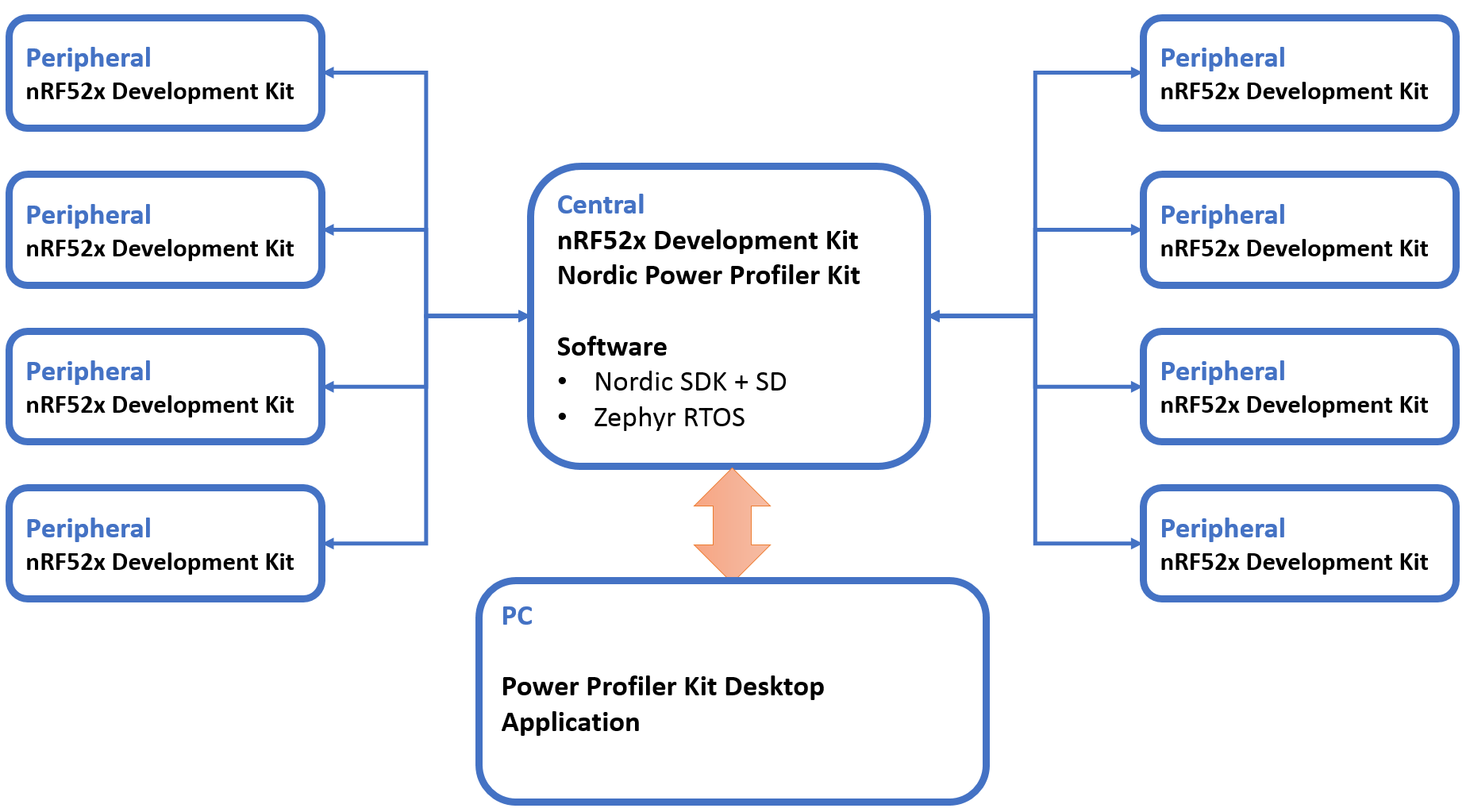


Figure :Behaviour measurement on BLE Central Schema Block

Then, the following performance are measured:

* **Power Consumption**
* **Interrupt latency**
* **Bluetooth Low Energy performance**

Finally, the measurements are compared. The results must not define which system is the best. This is only a representation of the performance of Zephyr RTOS and if this system is a good solution for tiny embedded system using Bluetooth Low Energy.

Note: A definition of the SoftDevice and Software Development Kit of Nordic can be found at **6.1 Environment Layer**.

# Bluetooth Low Energy

This chapter only gives a small introduction of the Bluetooth Low Energy. Therefore, the explanations are really simplified.

## Introduction

The Bluetooth Low Energy (BLE) or Bluetooth smart was introduced as part of the Bluetooth 4.0 core specification in 2010. While there is some overlap with classic Bluetooth, there is a difference between Bluetooth and Bluetooth Low Energy.

Like Bluetooth, BLE operates in the 2.4GHz ISM band. However, in contrast to classic Bluetooth, BLE is designed to provide significantly lower power consumption by remaining sleeping except for when a connection event is initiated. The connection event is explained at chapter **4.3.2.1 Connection Interval**.

On the other hand, Bluetooth can handle a lot of data, e.g. video or audio, but consumes battery life quickly and costs a lot more. BLE is used for applications that do not need to exchange large amounts of data, e.g. sensors, and can therefore run on battery power for years at a cheaper cost.



Figure : Bluetooth Logo

The BLE core 4.0 specification can be download at <https://www.google.no/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&ved=0ahUKEwjY44ioy8XVAhXCb1AKHROJBYYQFggnMAA&url=https%3A%2F%2Fwww.bluetooth.org%2Fdocman%2Fhandlers%2Fdownloaddoc.ashx%3Fdoc_id%3D229737&usg=AFQjCNFY1IFeFAAWwimnoaWMsIRZQvPDSw>

## Roles

The Bluetooth Low Energy can be used in four different roles. One device may support multiple roles.

* **Broadcaster**, transmitter only
* **Observer**, receiver only
* **Peripheral,** usually slave / GATT Server
* **Central**, usually master / GATT Client

The Broadcaster and observer are independent. However, the peripheral and the central are bonded each other. It is always the central that initiates connection to peripherals and a central can support multiple connections. But the peripheral can only be connected to one central.

In this document, the peripheral is considered as a server and the central as a client which is the most current case.

## Connection

The BLE connection between a central and a peripheral is separated in three parts.

* **Advertising** (peripheral) and **scanning** (central)
* **Connection procedure** and **services discovery**
* **Connection event**

### Advertising and Scanning

The advertising is performed by the peripheral. It consists to send information about the peripheral to any device that want to listen.

The scanning is performed by the central. It consists to listen the information sent by a peripheral. If the information satisfied the central, it runs a connection procedure with the peripheral.

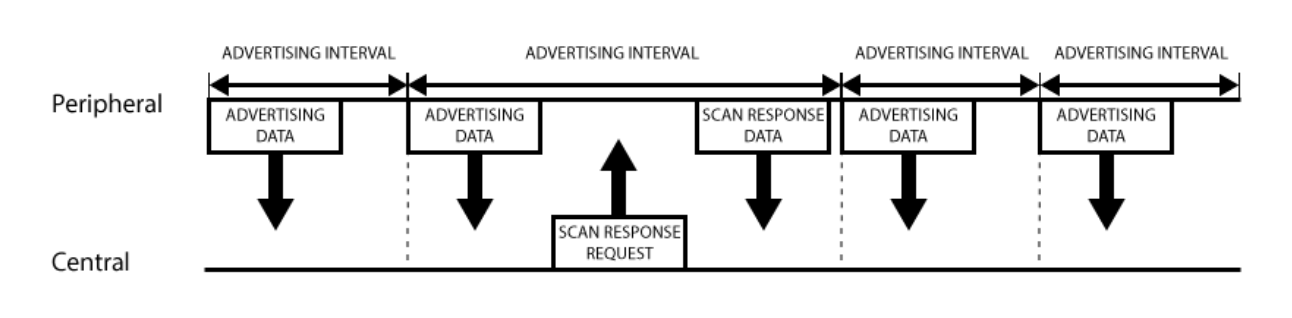


Figure : Scanning and advertising procedure

### Connection Parameters

In a BLE connection, four parameters handle the connection:

* **Connection interval min**
* **Connection interval max**
* **Connection Timeout**
* **Slave Latency**

Those parameters are why the BLE is low power consumption and are always defined by the central. But the peripheral can suggest to the central parameters that are more suitable for him.

#### Connection Interval

Data are transmit only on connection events. Those connection events are sent by the central. The peripheral can only send data on a connection events.

#### Connection Timeout

However, the central must send a connection event and a peripheral must response otherwise a timer is enable. When the timeout is passed and no other response from one of the devices, the link is considered lost and the devices close the link.

#### Slave Latency

On the other hand, the goal of the BLE is to be low power consumption. But it is a loss of power if the peripheral must response to the central and it has even no data. The Slave latency allows the peripheral to not response to several connection events to save batteries.

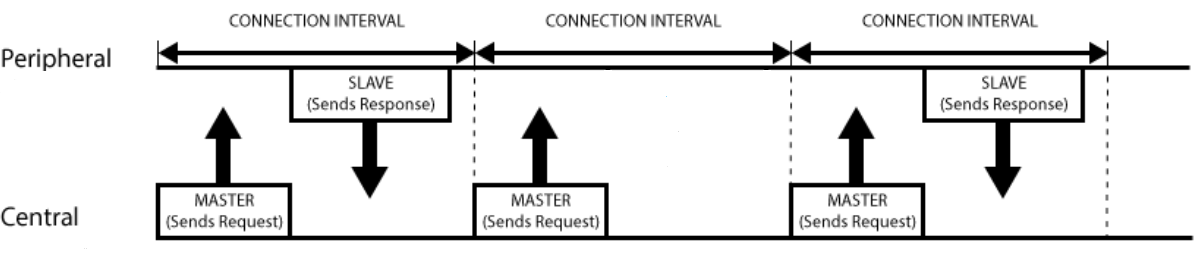


Figure : Connection events with slave latency of 1

## Profiles, Services and Characteristics

All profiles, services and characteristics are stored in the peripheral, called GATT server as well. Then, the central sends request to get information or value about them.

### General Definitions

#### Bluetooth SIG

Bluetooth Special Interest Group (SIG) is the [body](https://en.wikipedia.org/wiki/Standards_organisation) that oversees the development of [Bluetooth](https://en.wikipedia.org/wiki/Bluetooth) [standards](https://en.wikipedia.org/wiki/Standardization) and the licensing of the Bluetooth technologies.

#### UUID

The Universally Unique Identifier (UUID) is a 128bits unique number attributed to:

* **Type of an attribute**
* **Services**, which is contained in the attribute value
* **Characteristics**, which is contained in the attribute value
* **Descriptors**, which is contained in the attribute value

Some UUID are already defined by the Bluetooth SIG. Those UUID are only represented by 16bits.

#### Handle

The attribute handle is a unique 16-bit identifier for each attribute on a peripheral server. The central use this value to access to an attribute and not the UUID.

#### Type

The attribute type is a UUID that define the kind of data present in the attribute. Typical types are:

* **Service declaration**
* **Characteristic declaration**
* **Characteristic value**
* **Characteristic descriptor**

#### Permission

The attribute permission specifies the operations allows on the attribute and the security requirements. Typical operations are:

* **Read**
* **Write**

Attention: If a characteristic allows the central to read it value and the attribute permission read is not enabled, the central is not able to read the value.

#### Attributes

An attribute is the smallest entity used to define services and characteristics by the GATT Server. An attribute has:

* **Handle**
* **Type**
* **Permission**
* **Value**, which is a UUID, a sensor value, possible operations, …

The attributes are organized in an array and the handles are the number of the rows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Handle** | **Type** | **Permission** | **Value** |
| Service Declaration | 0x0000 |  |  |  |
| Characteristic Declaration | 0x0001 |  |  |  |
| Characteristic Value | 0x0002 |  |  |  |
| Characteristic Descriptor | 0x0003 |  |  |  |
| Characteristic Declaration | 0x0004 |  |  |  |
| Characteristic Value | 0x0005 |  |  |  |
| Service Declaration | 0x0006 |  |  |  |
| Characteristic Declaration | 0x0007 |  |  |  |
| Characteristic Value | 0x0008 |  |  |  |
| Characteristic Descriptor | 0x0009 |  |  |  |
|  |  |  |  |  |

Table : example of Attributes defined in a server

### Profiles

Profiles are definitions of applications and specify general behaviours that Bluetooth® enabled devices use to communicate with other Bluetooth devices. With BLE, the profiles are defined in the peripheral server. A Profile is composed of several services.



Figure : Profile

### Services

Services are the part of the profile that define an application performed by the profile. Some services are already defined to easier developers to make applications and firmware compatible. Typical BLE services are:

* [Current Time Service](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.current_time.xml)
* [Battery Service](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.battery_service.xml)
* [Blood Pressure](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.blood_pressure.xml)
* [Continuous Glucose Monitoring](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.continuous_glucose_monitoring.xml)
* [Heart Rate](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.service.heart_rate.xml)
* …

A list of defined BLE profile can be found at <https://www.bluetooth.com/specifications/gatt>. Those services are defined by the Bluetooth (SIG). However, it is possible to create your own services.

The attributes contain within the services are:

* **Service declaration,** which contains the service UUID
* **The attributes of the characteristics**



Figure : Service

### Characteristics

the characteristic is a container for user data. As the services, some characteristics are already defined:

* [Date Time](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.date_time.xml)
* [Battery Level](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.battery_level.xml)
* [Blood Pressure Measurement](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.blood_pressure_measurement.xml)
* [Heart Rate Measurement](https://www.bluetooth.com/specifications/gatt/viewer?attributeXmlFile=org.bluetooth.characteristic.heart_rate_measurement.xml)
* …

The attributes contain within the characteristic are:

* **Characteristic declaration**, which provides metadata about the actual user data
* **characteristic value**, which is a full attribute that contains the user data in its value field
* **Descriptor** (optional),which further expand on the metadata contained in the characteristic declaration



Figure : Characteristic

#### Characteristic declaration

The attributes characteristic declaration gives three information about the characteristic:

* **UUID of the characteristic**
* **Value handle**, handle of the attribute characteristic value
* **Properties**, operation permitted on the characteristic

Usually, the properties of the characteristic describe how the central accesses to the characteristic’s data. The main properties are:

* **Read**, the central reads from the value of the characteristic
* **Write**, the central writes to the value of the characteristic
* **Notify**, the central receives automatically the new value of the characteristic
* **Indicate**, the central receives automatically the new value of the characteristic and sends a confirmation.

The operation Notify and Indicate are enabled by the central using the attribute descriptor. Then the peripheral collects the data, when the function Notify or Indicate are called, and sends the data each connection event.

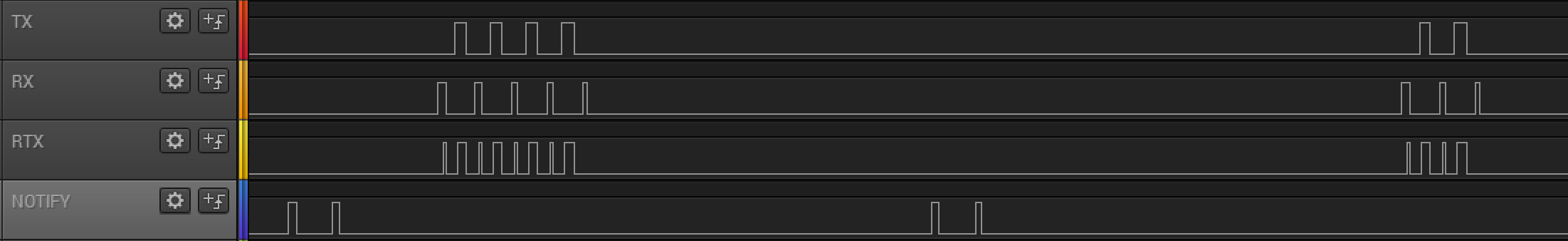


Figure : Notify measurement

#### Characteristic value

The characteristic value attribute contains the actual user data that the client can read from and write to for practical information exchanges. The value of a characteristic value attribute can contain any type of data imaginable.

#### Descriptor

The characteristic descriptor is used to provide the client additional information about the characteristic and its value.

The most important and commonly used is the Client Characteristic Configuration Descriptor (**CCCD**). The descriptor enabling or disabling the operation notify and indicate of the peripheral.

### Example

Each white square is an attribute.



Figure :Profile template

## BLE Stack

The Bluetooth Low Energy stack is separated in three parts:

* **Application**
* **Host,** which contains the configuration of device and the profiles and provides API for application
* **Controller,** which controls the connection requirements and the transceiver

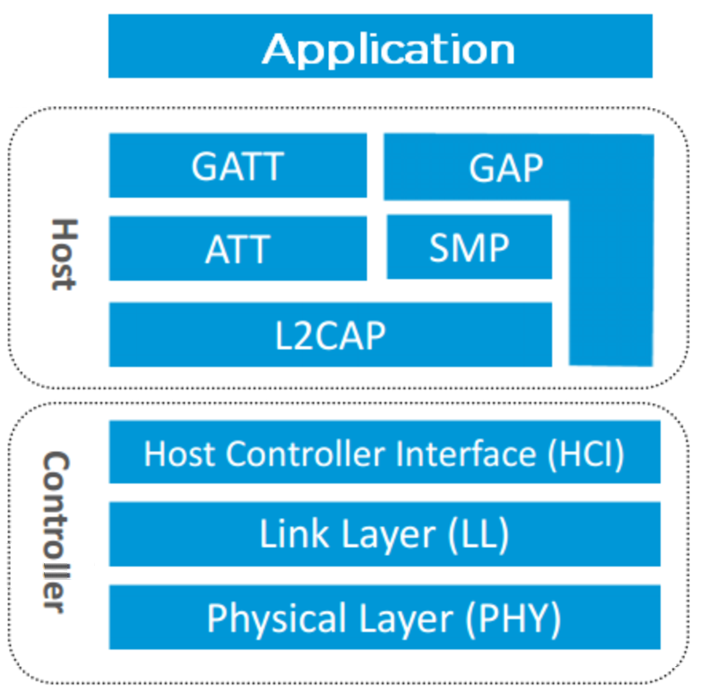


Figure :BLE Stack

### Host

**GAP**, General Access Profile, is in control of scanning, advertising, security establishment and connections. It defines the roles of the devices and additional data, e.g. its name.

**GATT**, General Attribute Profile, defines how data is organized and exchanged in between different applications. The data in GATT is organized in services and characteristics.

**ATT**, Attribute Protocol, defines the data in the form of attributes. This form is defined at chapter **4.4.1.6 Attributes**

**SMP**, Security Manager Protocol, offers the security procedure to a BLE application. For example, the typical security procedures are device authentication, device authorization, data integrity, data confidentiality.

**L2CAP**, Logical link control and adaptation protocol, prepares packets. If a packet from the upper layers is too large, this layer fragmented and recombines it for the transfer.

### Controller

**HCI**, Host Controller Interface, makes possible to interface a wide range of Hosts with the controller. Therefore, it is possible for a device to have several roles, Peripheral, Central, Broadcaster, Observer, at the same time.

**LL**, Link Layer, establishes and manages connections. It is in charge to send packets and to keep responding to or sending connection events at each connection interval.

**PHY**, Physical Layer, contains the analog communications circuitry used for modulating and demodulating analog signals and transforming them into digital symbols.

## BLE Requirement

### Priority

It is important that no connection events are missed. Hence, the BLE stack is high priority to no be suspended by any interrupt hardware or software.

### Number of connections

In the case of the role of the device is peripheral, the Peripheral is limited to be connected to only one central.

In the case of the role of the device is central, the central is not limited for number of peripheral connected. However, the number of connection is limited by the environment used.

|  |  |  |
| --- | --- | --- |
|  | SoftDevice | Zephyr RTOS |
| Number Peripherals Connected | 8 | 64 |

Table : Number max of connections for central

### MTU

ATT Maximum Transmission Unit (MTU) is the maximum length of an ATT packet. An ATT packet contains the information of the Read, Write, Indicate, Notify request and response.

The ATT MTU is defined by the L2CAP layer and can be theoretically anywhere between 23Bytes and infinity. However, this size is limited by the environment used.

|  |  |  |
| --- | --- | --- |
|  | SoftDevice | Zephyr RTOS |
| RX MTU max [Bytes] | 251 | 1300 |
| TX MTU max [Bytes] | 251 | 2000 |

Table : MTU max

The MTU limits the size of the data contains in the attribute characteristic value.



Table : Attribute packet without authentication signature

However, the BLE packet structure has a payload of 33Bytes and each layer in the protocol stack takes cut. Hence the ATT protocol has 23Bytes left, the minimal MTU, and 20Bytes for data (MTU – opcode - handle).

It is why, the L2CAP layer fragments and recombines ATT packets to send the ATT packet with several BLE packet at each connection interval.

### Time Requirement

The connection, advertising and scanning parameters are defined in a specific range of value.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Min | Max | Interval | Defined By |
| Connection Interval Min/Max | 7.5 [ms] | 4 [s] | 1.25 [ms] | Central |
| Connection Timeout | 100 [ms] | 32 [s] | 10 [ms] | Central |
| Slave Latency | 0 | 499 |  | Central |
| Scanning Interval | 2.5 [ms] | 10.24 [s] | 0.625 [ms] | Central |
| Advertising Interval | 20 [ms] | 10.24 [s] | 0.625 [ms] | Peripheral |

Table : Parameters value

Each BLE application require different connection parameters. The measurements performed in this project are adapted to the typical requirement.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Conn. Interval Min | Conn. Interval Max | Slave Latency | Conn. Timeout |
| Mouses | 7.5 [ms] | 15 [ms] | 20 | 3 [s] |
| Keyboard | 7.5 [ms] | 30 [ms] | 6 | 430 [ms] |
| Glucose Meter | 10 [ms] | 100 [ms] | 0 | 4 [s] |
| Power Profiling | 20 [ms] | 100 [ms] | 0 | 4 [s] |
| Heart Rate | 400 [ms] | 650 [ms] | 0 | 4 [s] |
| Blood Pressure | 500 [ms] | 1 [s] | 0 | 4 [s] |
| Current Time | 500 [ms] | 1 [s] | 0 | 4 [s] |
| Health Thermometer | 500 [ms] | 1 [s] | 0 | 4 [s] |

Table :Time requirements for typical use cases

Note: The values are the values indicated in the examples of the nRF5 SDK v13.0.0.

## Sources

<https://learn.adafruit.com/introduction-to-bluetooth-low-energy/introduction>

<https://www.safaribooksonline.com/library/view/getting-started-with/9781491900550/ch04.html>

<https://learn.mikroe.com/bluetooth-low-energy-part-1-introduction-ble/>

<https://www.link-labs.com/blog/bluetooth-vs-bluetooth-low-energy>

<https://www.bluetooth.com/specifications/profiles-overview>

<https://www.bluetooth.com/specifications/gatt>

<http://infocenter.nordicsemi.com/pdf/S132_SDS_v5.0.pdf>

<https://en.wikipedia.org/wiki/Bluetooth_Special_Interest_Group>

# Hardware

The Hardware is separate in six parts:

* **nRF52840 SoC**
* **Micro USB-B** to communicate with a PC and programme the chip
* **Power supply** provided by the Micro USB-B
* **Nordic Power Profiler** **Kit (PPK)** to measure the power consumption of the chip only
* **Connector interface** to connect the extension board, Power Profiler Kit and nRF52840 DK
* **Extension Board** to provide data to the chip

The extension board is the only part that is developed for this project. The Power supply 3V, the Micro USB-B and the nRF52840 SoC are on the nRF52840 Development Kit.



Figure : Hardware schema block

## nRF52840 SoC

The SoC used is the nRF52840. Here is the description from Nordic Semiconductor website.

*“The nRF52840 is an advanced multi-protocol SoC ideally suited for ultra-low power wireless applications. The nRF52840 SoC is built around a 32-bit ARM® Cortex™-M4F CPU with 1MB flash and 256kB RAM on chip. The embedded 2.4GHz transceiver supports Bluetooth® low energy (*[*Bluetooth 5*](https://www.nordicsemi.com/eng/Products/Bluetooth-5)*),* [*802.15.4*](https://www.nordicsemi.com/eng/Products/IEEE-802.15.4-Thread)*, ANT and proprietary 2.4GHz protocols. It is on-air compatible with existing nRF52 Series, nRF51 Series, and nRF24 Series products from Nordic Semiconductor.”*

In addition of the description above, those features are used for the project:

* **1.7 to 5.5V** power supply
* **80mA** current consumption max
* **PPI** – Programmable Peripheral Interconnect
* **48 x GPIOs** (32 x GPIOs PORT0, 16 x GPIOs PORT1)
* **2 x I2C** (100kHz, 250kHz, 400kHz)
* **4 x SPI** (125kHz, 250kHz, 500kHz, 1MHz, 2MHz, 4MHz, 8MHz, 16MHz, 32MHz)

There no important reason to use this processor. But the nRF52840 is the SoC that provides the largest number of features. Hence, it will avoid being limited with the measurements because of the capacity of the SoC.

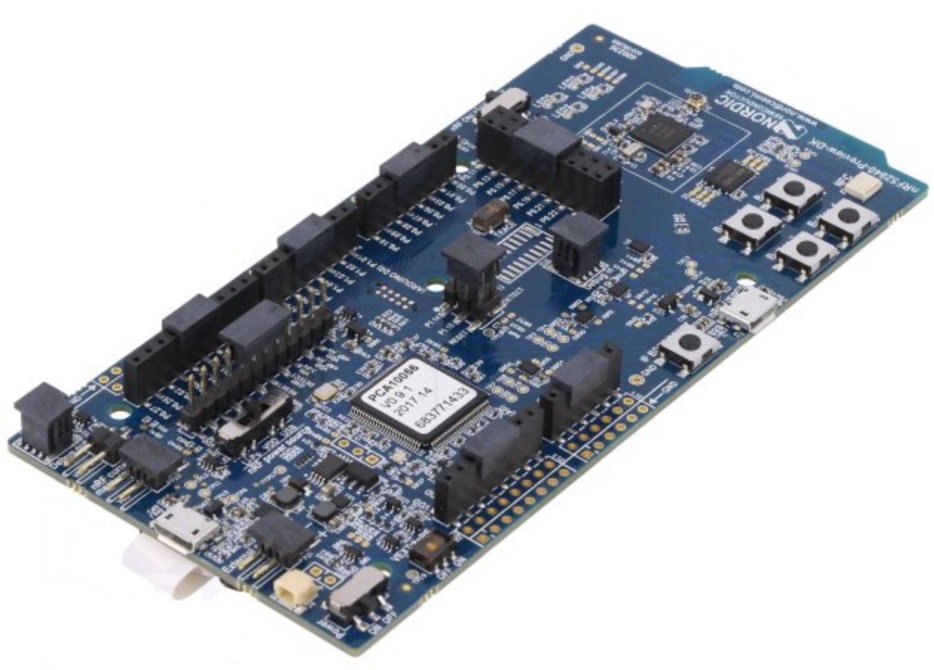


Figure : Nordic nRF52840 Preview Kit

## Micro USB-B

The connector Micro USB-B is used for different purposes:

* **To programme** the nRF52840 Chip
* **To provide 5V power supply** to the system
* **To Debug** the nRF52840 Chip
* **To transfer the data measured** with SEGGER Real Time Transfer

The connection SEGGER Real Time Transfer (RTT) is used by the Nordic Power Profiler Kit to transfer the data to a PC.

## Power Supply

The power supply transforms the power supply of 5V, provided by the Micro USB-B, to 3V to supply all the system. To transforms the power supply, there is a fixed 3V buck regulator and one voltage follower regulator on the nRF52840 DK.

Due to the low consumption of the system, the power supply of the nRF52840 DK is far enough to provide power to all the system.



Figure : Regulator buck 3V schematic, from nRF52840 DK User Guide

## Connector Interface

The connector interface is defined by the nRF52840 development kit and is almost the same for all the Nordic’s DK.

The connector allows:

* **To supply the extension board and the Power Profiler Kit**
* **To measure the power consumption** of a nRF5x SoC
* **To access to the GPIOs** of a nRF5x SoC

The extension board and the Power Profiler Kit are plugged on the nRF52840 DK with the connector interface that allow an easy connection of the different part of the system.



Figure : nRF52840 Preview DK board connectors, from nRF52840 DK User Guide

## Nordic Power Profiler Kit

The Nordic Power Profiler Kit is an easy use tool for the measurement and power consumption optimization of embedded solutions. It provides the following features:

* **1µA-70mA current measurement range**
* **0.2µA measurement resolution**
* **77kHz sampling rate**
* **Desktop application** in python allowing customization

The Nordic Power Profiler Kit is the best solution because it allows to measure only the power consumption of the nRF5x chip.

The Desktop application communicate with the PPK using the SEGGER Real Time Transfer of the nRF5x Chip. This connection can be used to transmit other measurements via RTT. The Desktop application can be modified to display those measurements.



Figure : Nordic Power Profiler Kit view

## Extension Board

The extension board is the single part that the hardware is developed. It is connected to the nRF52840 DK using the interface connector.

The purpose of the extension board is to use the nRF52840 Chip under real conditions of use. To do that, the extension board provide the different elements.

* **A/D converter** that can be connected to a generator function or an external sensor.
* **Accelerometer**
* **Interrupt generator**

All components as some general criterion to ease the order:

* **Package easy to solder**
* **Same provider**

All the hardware, schematic and PCB, of the extension board is developed with the Software [Altium Designer 17™](http://www.altium.com/altium-designer/whats-new).

### A/D Converter

The A/D Converter provides a large quantity of data that the chip must be able to deal with no loss. As the large quantity of data to stress the chip, the way to get the data must stressful as well.

The component used is the Delta-Sigma ADC **MAX11200** that provides the following features:

* **24 bits Resolution and 8 bits register’s address**
* **2.7 to 3.6V** power supply
* **300µA** currentconsumption max
* **Reference Voltage**
* **SPI (SCL max 5MHz)** to calibrate and get the data
* **Ultra-low-power** with power-down mode

An ADC communicating with SPI is chosen due to the requirement to communicate fast.

The Analog input can be provided by a function generator (Connector BNC 50Ω) or by an external analogue sensor (Pin 2x1). Two resistances can be soldered to use the ADC in current loop system.

The Reference voltage can be the power supply voltage or an external reference if the external sensor has specific requirement.

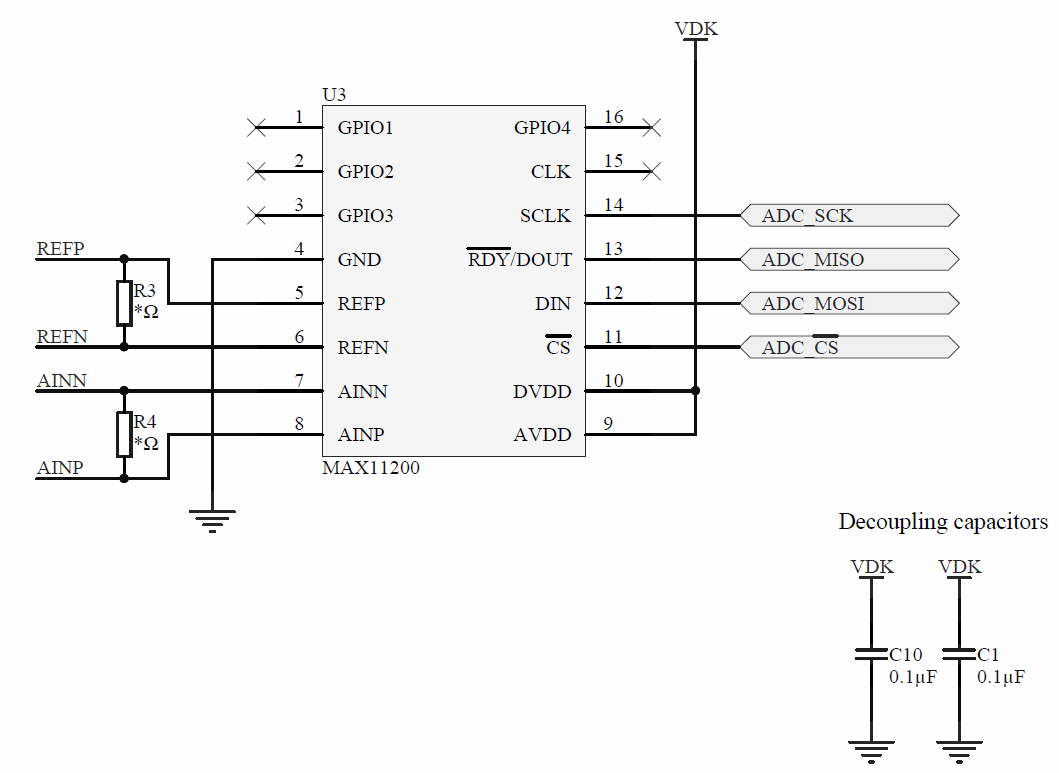


Figure :MAX11200 schematic

#### *Throughput*

The A/D Converter can be used in Continuous Conversion or Single -Cycle Conversion. For each mode, an oscillator intern can be selected, 2.4576MHz or 2.048MHz, to determine the data rate. Only the Continuous conversion is used.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Oscillator |  |  |  |  |  |
| 2.4576MHz | 60 | 120 | 240 | 480 | [sps] |
| 2.048MHz | 50 | 100 | 200 | 400 | [sps] |

Table : A/D Converter Data Rate in sample per second

The resolution of the A/D Converter is 24bits and the register address size is 8bits.

The frequency max of the SPI is 5MHz but due to the SPI frequency provided by the nRF52840, the frequency max of the SPI is 4MHz.

The throughput of the A/D Converter is interesting because it is near to the minimum connection interval of the BLE.

### Accelerometer

As the A/D Converter, the accelerometer provides a large quantity of data that the chip must be able to deal with no loss.

The component used is the accelerometer **LIS3DH** that provides the following features:

* **16 bits Resolution**
* **3-axis**
* **±2g/±4g/±8g/±16g**
* **1.7 to 3.6V** power supply
* **185µA** currentconsumption max
* **FIFO 32-level 6 bytes**
* **I2C (SCL max 400kHz)/SPI (SCL max 10MHz)** to get the data
* **2 Interrupt pins** to notify when new data are available
* **Ultra-low-power** with automatic power-down mode

An accelerometer communicating with I2C is chosen to use different features of the chip. However, the SPI can be used as well.

The LIS3DH provide a FIFO to store data. This FIFO can be read at one time with a frame of 192 bytes, 2 bytes per axis. This frame provides a good test to stress the chip.

Other features of the LIS3DH are the Click and Free-fall detection that generate interruption. Those features are interesting because they can simulate real interruption from a sensor.

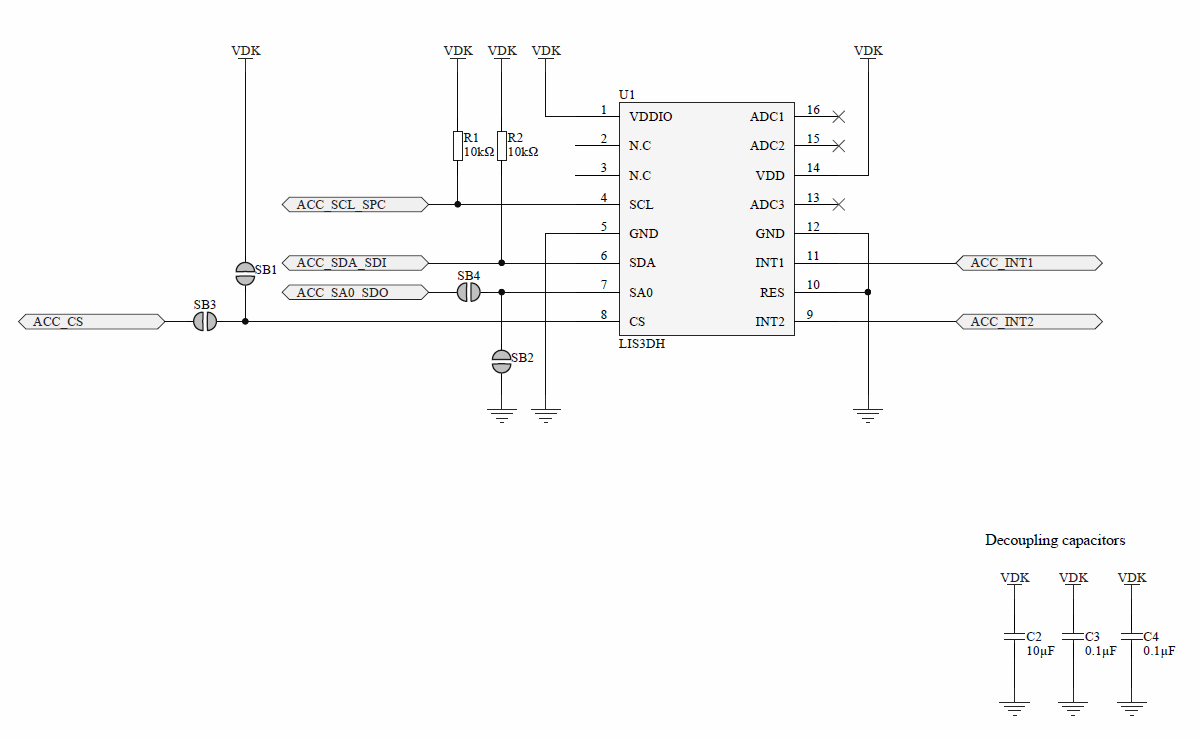


Figure :LIS3DH Schematic

#### *Throughput*

The accelerometer can perform measurements in different mode. For each mode, the data rate can slightly change. The LIS3DH provides a large range of data rate.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Mode |  |  |  |  |  |  |  |  |  |  |
| Low Power | 1 | 10 | 25 | 50 | 100 | 200 | 400 | 1.6k | 5.376k | [Hz] |
| Normal | 1 | 10 | 25 | 50 | 100 | 200 | 400 | 1.344k |  | [Hz] |
| High Resolution | 1 | 10 | 25 | 50 | 100 | 200 | 400 | 1.344k |  | [Hz] |

Table : Accelerometer Data Rate in Hertz

The resolution depends of the mode but it has no influence on the data frame.

* Low Power: 8bits
* Normal: 10bits
* High Power: 12bits

However, the accelerometer has a 32bytes FIFO for each byte of measurement, axis LSByte and axis MSByte. The register address size is 8 bits and the I2C requires 9 bits. More, the I2C requires a ACK bit between each byte transmitted.

The minimum data frame is calculated with the data of one axis. Then, the frequency max of the I2C is 400kHz.

The throughput of the Accelerometer is interesting due to the FIFO that create a long data frame and because it is near to the minimum connection interval of the BLE.

### Interrupt generator

The interrupt generator generates pulse that create interruptions in the programme. The period of interruptions can be easily changed to modify the test conditions.

The component used is the Programmable Waveform Generator **AD9837** that provides the following features:

* **16MHz Clock**
* **28 bits (0.06Hz)** **Resolutions**
* **2.3V to 5.5V power supply**
* **4.5mA** currentconsumption max
* **3 Wires SPI** to programme the waveform type and frequency
* **Low power** with power-down option

A 28Bits Register is used and programmable via SPI to calculate the frequency. The formula below defines the frequency:

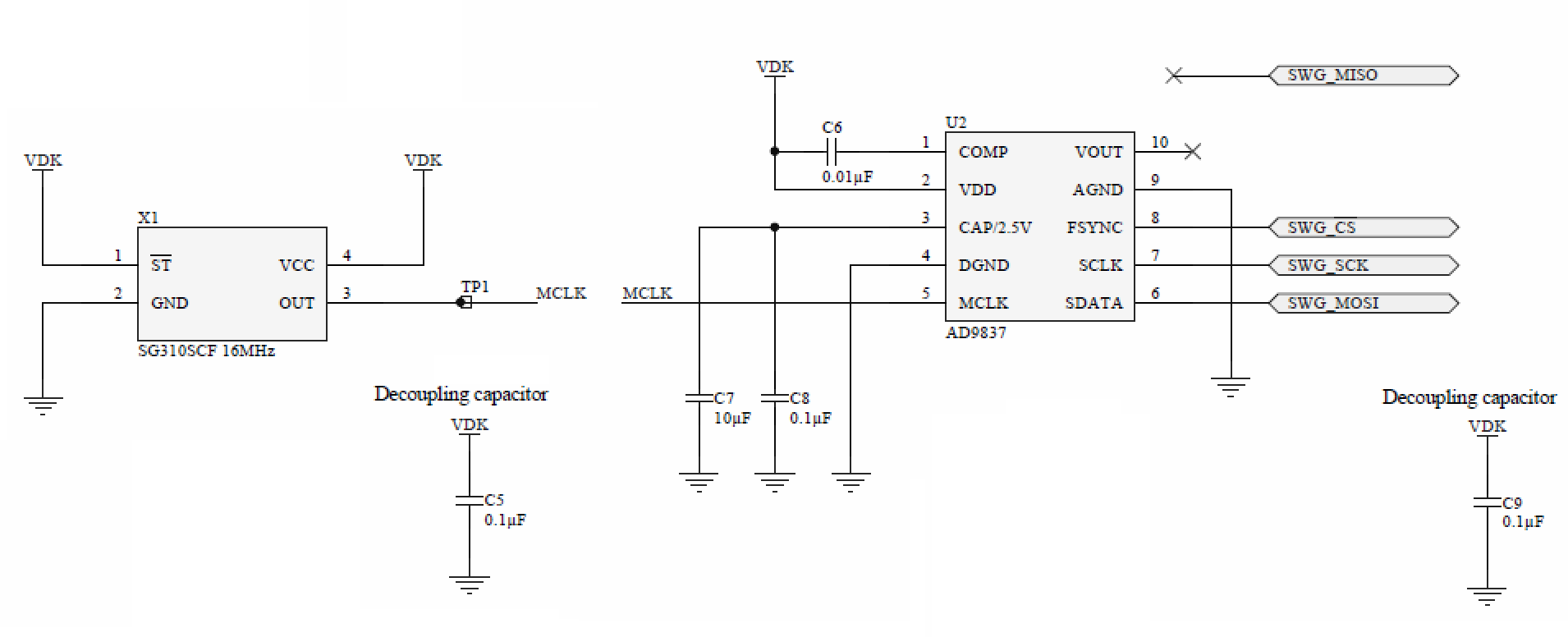


Figure : AD9837 schematics

## Annexes

* List of components
* Schematic
* PCB
* Pin Map

## Sources

<https://www.nordicsemi.com/eng/Products/nRF52840>

<https://www.nordicsemi.com/eng/Products/nRF52840-Preview-DK>

<https://datasheets.maximintegrated.com/en/ds/MAX11200-MAX11210.pdf>

<http://www.st.com/content/ccc/resource/technical/document/datasheet/3c/ae/50/85/d6/b1/46/fe/CD00274221.pdf/files/CD00274221.pdf/jcr:content/translations/en.CD00274221.pdf>

<http://www.analog.com/media/en/technical-documentation/data-sheets/AD9837.PDF>

# Measurements

The measurements performed are:

* Interrupt Latency
* Power Consumption
* BLE Radio state
* BLE Application → Controller stack propagation

To perform the measurement, it is important that the code does not perturb the application in anyway. To limit the perturbation, the measurement code uses directly the registers of the nRF52840 without any libraries.

## Interrupt Latency

### Why

The interrupt latency represents the time it takes to the interruption to be serviced. As the response time of real-time systems is important, the interrupt latency determines the minimum interval between interruptions to correctly handle them. Hence, the interrupt latency must be as low as possible.

There are two types of interrupts:

* **Hardware**, interrupts are triggered by peripheral devices outside the microcontroller
* **Software**, interrupts are an instruction which cause a switching context to an interrupt handler like a hardware interrupt.

Both types of interrupts have different sources that can contribute to the interrupt latency. The main sources come from RTOS:

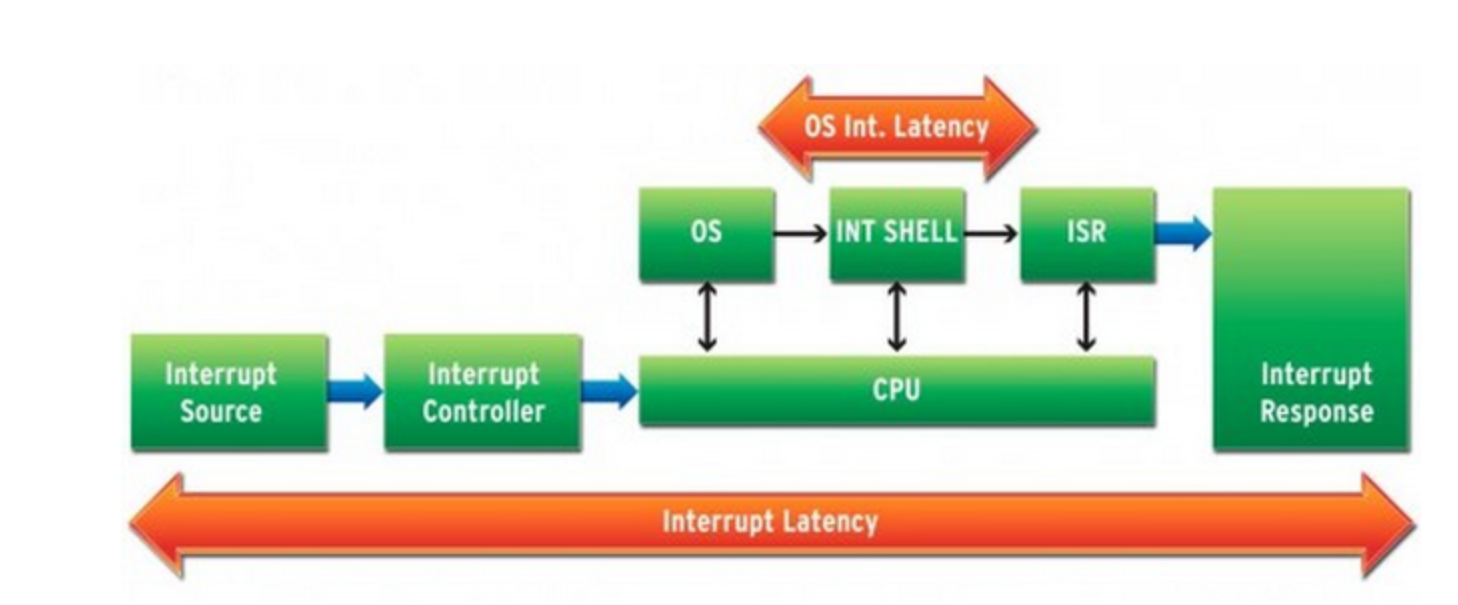
* When an RTOS disables interrupts while accessing critical OS data.
* When the context, e.g. status registers, is saved and resorted before and after processing the interrupt.
* When a context switch occurs to defer processing and to return to an RTOS task or threads.
* When an ISR interact with an RTOS by making system call such as semaphore.

However, the architecture of the CPU influences the interrupt latency as well. The interrupt latency of the CPU is constant and specified by the manufacturer and correspond to the interrupt entry.

The interrupt latency for interrupt entry is the number of processor clock cycles between an interrupt signal arriving at the processor and the processor executing the first instruction of the interrupt handler.

The nRF52840 is built around a 32-bit ARM® Cortex™-M4F CPU. Its interrupt latency on entry is 12 cycles, plus a possible additional 17 cycles for Cortex-M4 with Floating Point Unit.

The clock of the nRF52840 used is 64MHz, the interrupt latency converted in time is:



MEILLEUR IMAGE

### Expected Results

First, the interrupt latency should not depend of the frequency of the interruptions. However, it could increase when several interrupts occur in the same time.

Secondly, the priority of the Bluetooth Controller is higher than any interrupt. Hence, the interrupt latency should be the highest when the Bluetooth Controller is sending or responding to a connection event.

### How

The easier way to measure the interrupt latency is to toggle a pin on a GPIO interface. Using the GPIO allows to not disturb the system and to have a slight error. However, it is important to measure this error.

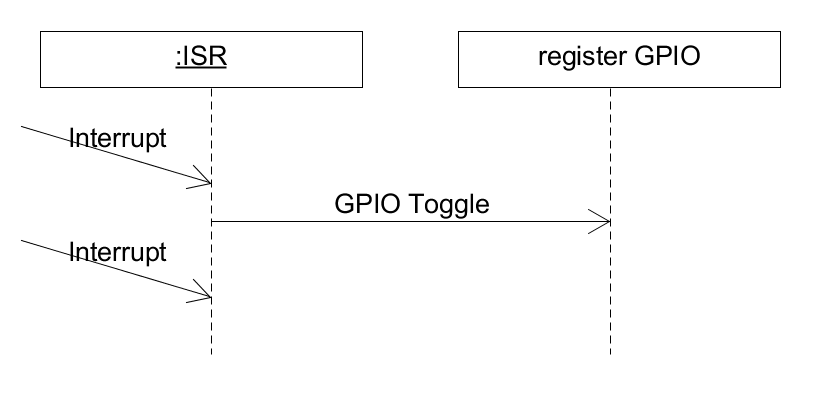


Figure 27: Sequence to measure the interrupt latency

Two interrupts latency are measured:

* **GPIO** Interrupt latency
* **Software Timer** interrupt latency

#### GPIO Interrupt Latency

The GPIO Interrupt Latency is a measurement from a peripheral, hardware interrupt latency.

An input pin on a GPIO interface generates the interrupt and another output pin is toggled in the ISR. Then, the interval between the two signals is measured.

#### Software Timer Interrupt Latency

The Software Time interrupt Latency is a measurement from an instruction, software interrupt latency.

An output pin is toggled in the timer ISR. The interval between each edge is measured and as the period of the timer is known, the interrupt latency can be calculated.

## Power Consumption

### Why

The power consumption defines the autonomy of an embedded system. Longer the autonomy is, better is the embedded system. Hence, the power consumption must be as low as possible.

### Expected Results

As Zephyr is an operating system, it Is expected that it power consumption is higher than the power consumption of the SD+SDK. Secondly, the power consumption will increase in the same time than the frequency of interruption.

### How

The power consumption is measured with Nordic Power Profiler Kit. However, to measure correctly the power consumption, it is important to disable useless drivers in the configuration to not impact the results. It means to disable for:

**SD+SDK**

* SEGGER RTT

And for **Zephyr RTOS**

* UART driver
* Console driver

Unfortunately, no output on console is possible during the power consumption test to measure as much as possible the current consumption in real cases.

## Bluetooth Low energy

It is important to note a great difference between the SoftDevice and Zephyr RTOS.

* SoftDevice is confidential and no access is possible
* Zephyr RTOS is an open source project, hence all the source code is accessible.

This difference is important because, to compare the behaviour, the same tests must be performed and it is not possible to measure the BLE performance directly within the stack. It why the number of tests possible is limited. However, it is possible to use different peripherals of the nRF52840 to get some information.

### BLE Radio State

#### Why

It is important that a BLE connection is not lost because one of the devices connected is not able to response correctly. Seeing the states of this radio allows to see if the device still responses and when it sends or receives a packet.

On the other hand, it allows to see the radio latency with the central and to see if the time requirement regards the connection events with peripherals are met.

### Expected Results

The stack Bluetooth is high priority. It means that no interrupt or any event should stop the capacity of the device to response to or send a connection event. Thus, it is expected that the connection is maintain even with a high frequency of interruptions.

#### How

The nRF52840’s Radio has a State Machine and the operating state of the Radio is control via several events and task.



Figure 28: Bluetooth radio state of nRF52840, from nRF52840 Datasheet

The nR52840 SoC provides registers named TASKS and EVENTS. A TASK register does some job like start or stop a module and an EVENT register is like a status register that indicate an event occurred. Then, the Radio state is measured using the following peripherals:

* **PPI**, Programmable Peripheral Interconnect, enables peripherals to interact autonomously and it eliminates the need for CPU activity.
* **GPIOTE**, GPIO Tasks and Events, provides functionality for accessing GPIO pins using TASKS and EVENTS registers.
* **RADIO** transceiver used for the BLE transmission.

The RADIO’s EVENTS are connected directly to the GPIOS using PPI. Three GPIOS are used to represent The RADIO’s states.

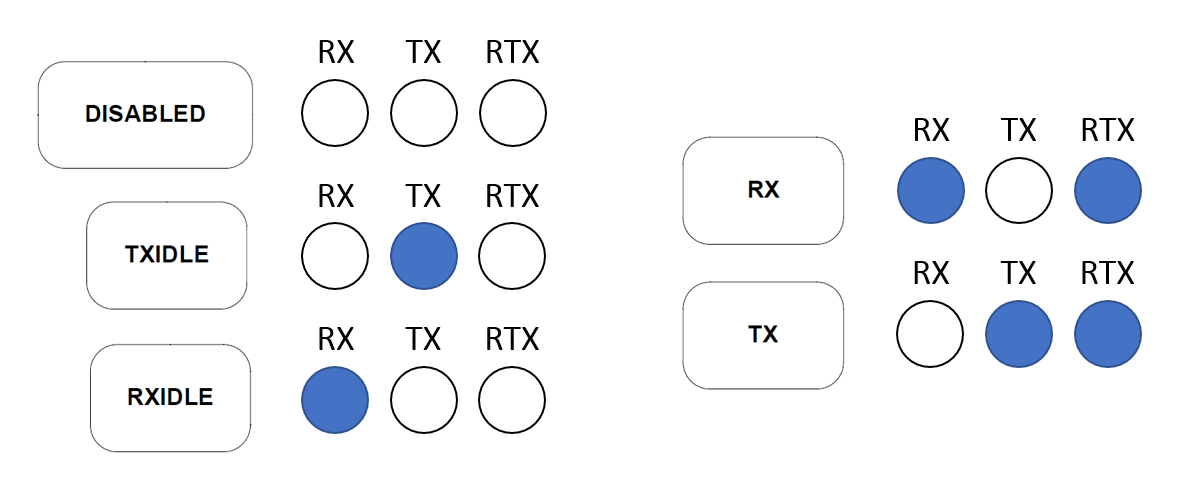


Figure 29:GPIOs enabled for each radio's states

Another EVENT, CRCOK, is used to see if the packet received was wrong. Finally, it is possible to see:

* When the device receives a packet
* When the device sends a packet
* When the CRC of the packet is correct

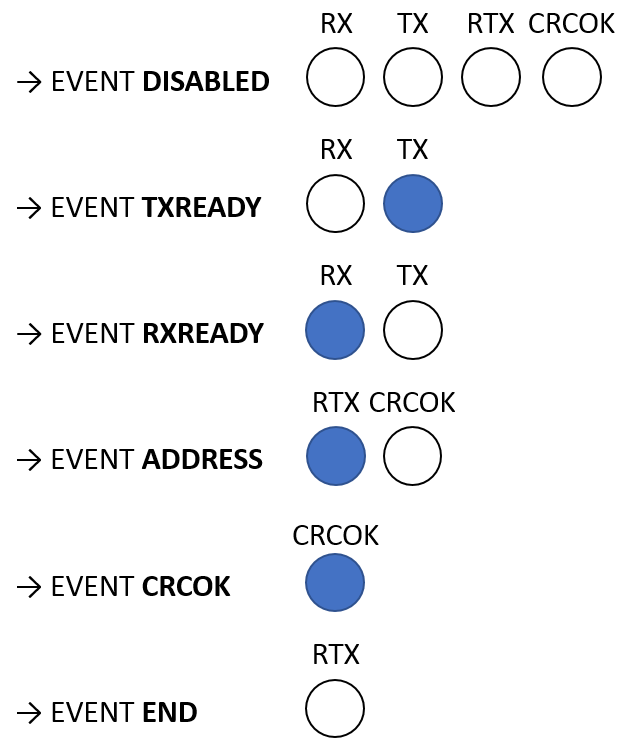


Figure 30: GPIOs enabled and disabled for each event

#### BLE Application → Controller Stack Propagation Delay

#### Why

The stack propagation delay is the time for the BLE driver to prepare data and push a packet within the BLE Link Layer TX Buffer. Measuring the stack propagation delay allows to see how much time a process is locked to send data. The stack propagation time must be as short as possible.

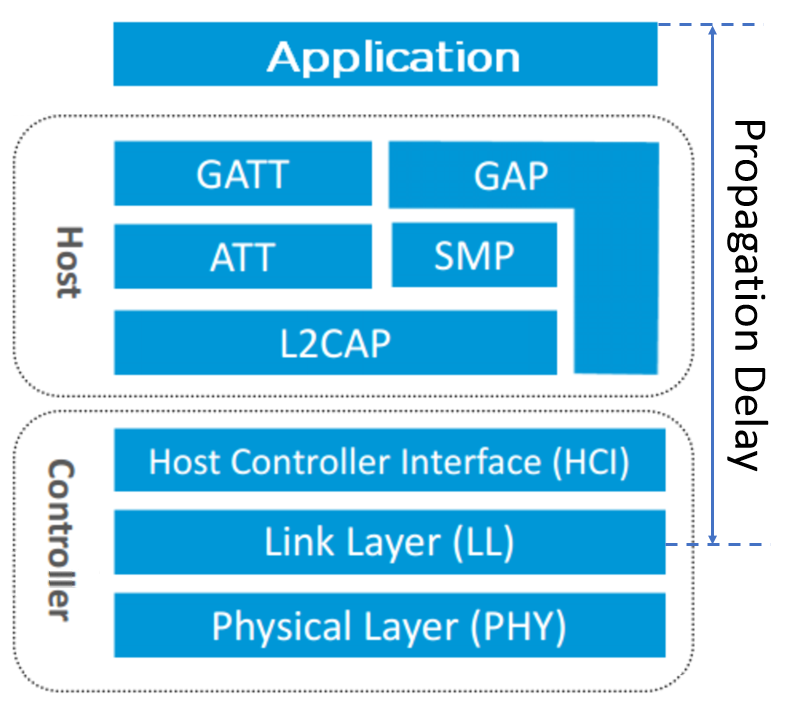


Figure 31: Layers include within the propagation delay

On the other hand, it allows to see if BLE driver is still able to handle packets without error. An error can occur when no more resources are available.

### Expected Results

It is expected that the stack propagation delay increase at the same time than the frequency of the interruptions. However, it should be still possible to push a packet without error.

#### How

The stack propagation latency is measured using the GPIO. Two GPIO are used:

* **ERROR**, if an error occurred when a packet is pushed
* **REQUEST**, if notify or indicate with peripheral or write, read and enable CCCD with central

The GPIOs is turned on right before pushing a packet and it is turned off right after. The figure below shows the process with the notification.

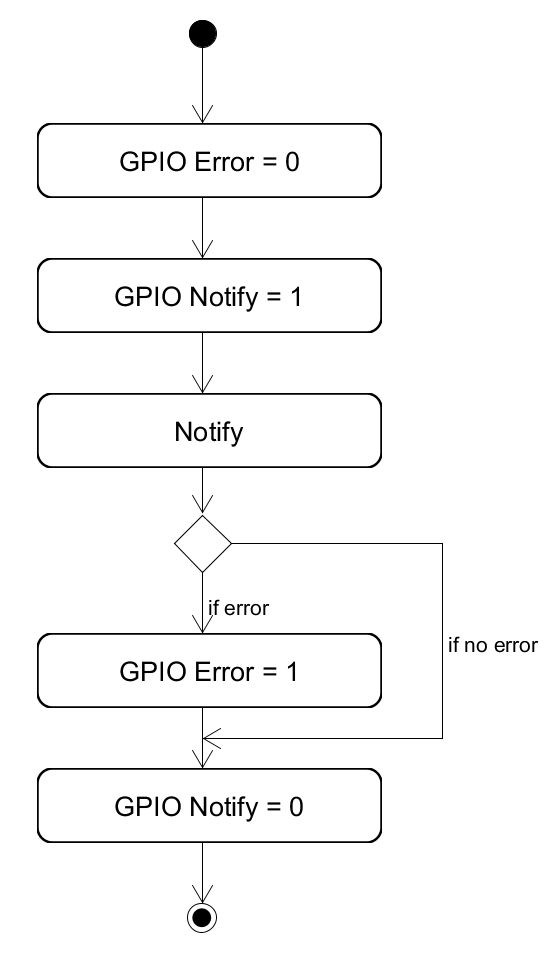


Figure 32: Measurement of stack propagation delay to notify

#### Other Interesting BLE Measurements

Because of a lake of time and the limitation regards the SoftDevice, some interesting measurements were not carried out:

* BLE Controller → stack propagation delay
* BLE radio interrupt latency
* BLE with encryption
* Integrity of the transmitted data

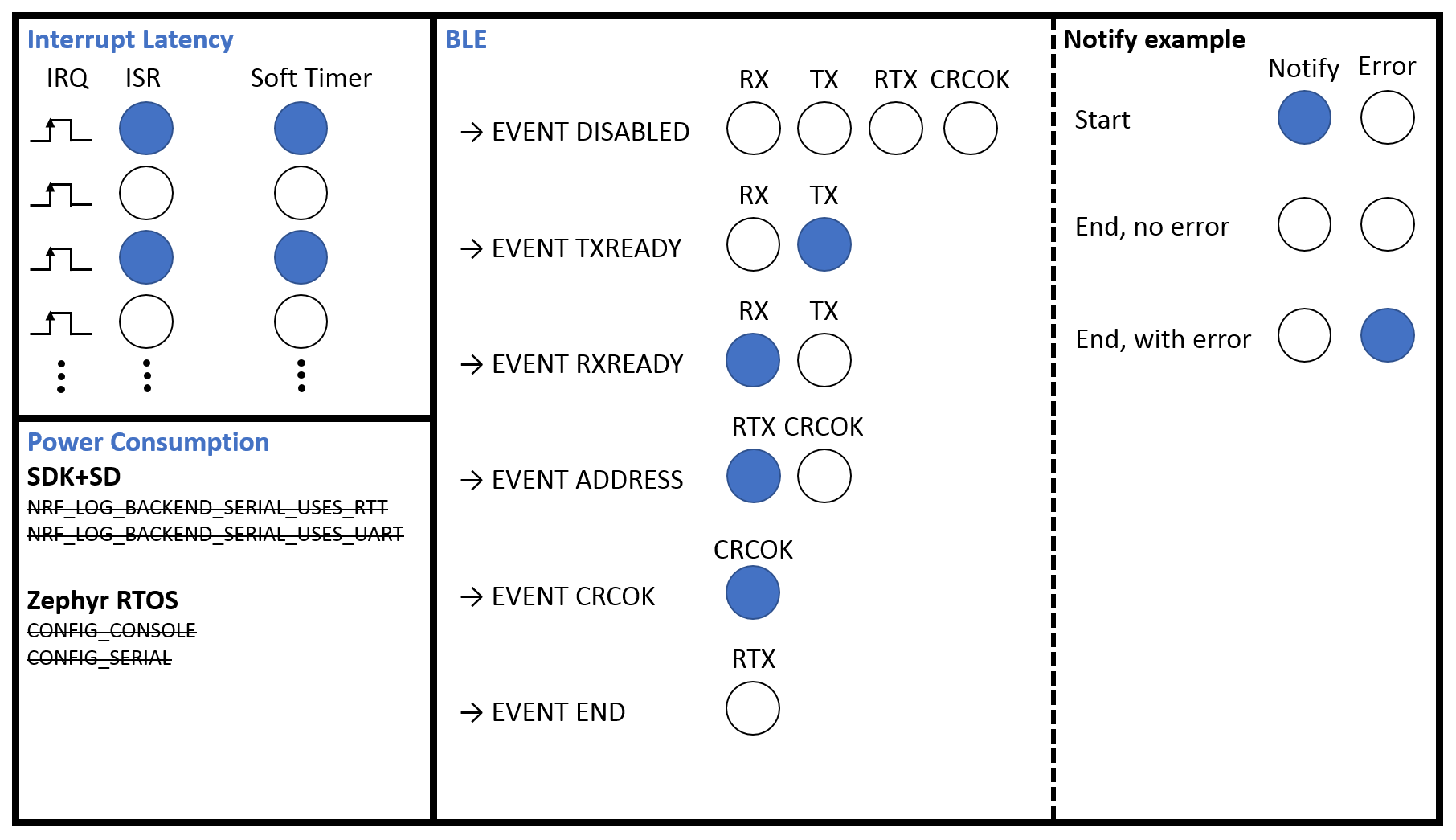


Figure 33: Measurements

## Configurations

The configurations explained concern the data rates and the BLE stack:

* **ACC Data Rate**: 200sps
* **ADC Data Rate**: 240sps
* **BLE TX Buffer**: 88 packets
* **BLE MTU**: 23Bytes

The BLE TX Buffer is calculated with the data rates of the drivers and a connection event of 200ms. 200ms is used to take a marge compared to the biggest use cases.

## Test Cases

## Annexes

## Sources

<http://infocenter.nordicsemi.com/pdf/nRF52840_OPS_v0.5.pdf>

<https://community.arm.com/processors/b/blog/posts/beginner-guide-on-interrupt-latency-and-interrupt-latency-of-the-arm-cortex-m-processors>

<http://www.bogotobogo.com/Embedded/hardware_interrupt_software_interrupt_latency_irq_vs_fiq.php>

<https://blogs.mentor.com/colinwalls/blog/2012/06/05/measuring-interrupt-latency/>

<http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.faqs/ka16366.html>

# Software

The Software is separated in four parts:

* **Environment Layer**
* **Abstract Layer**
* **Driver Layer**
* **Application Layer,** which is use all the below layer to create a peripheral and a central application.

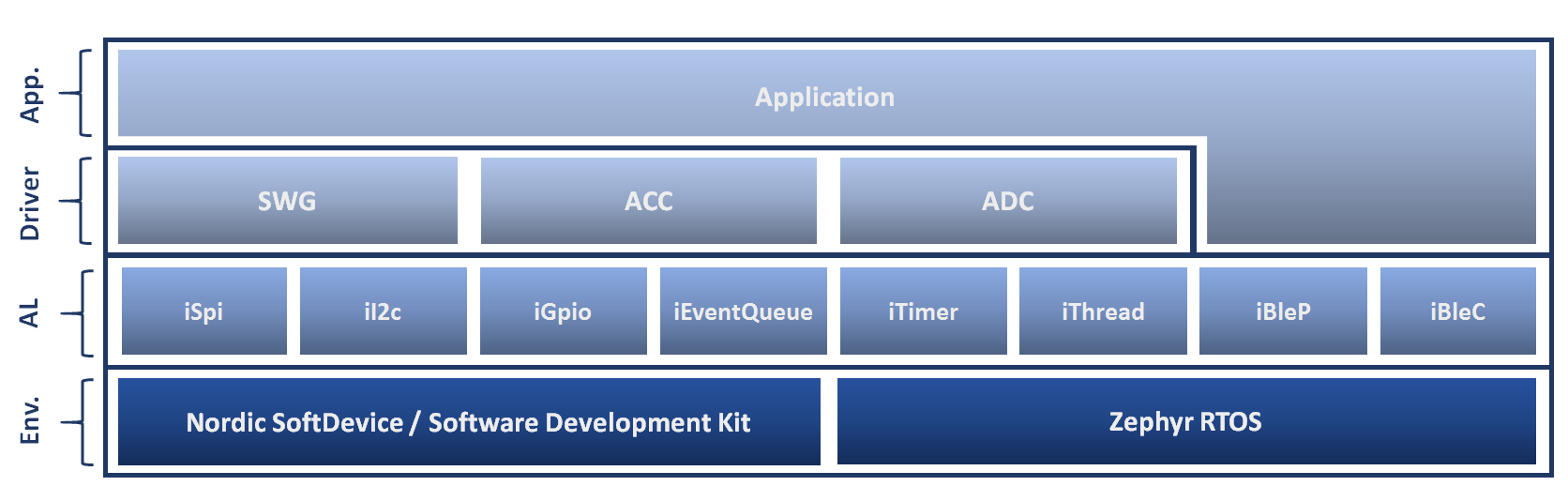


Figure : Architecture of the Software

## Environment Layer

The Environment Layer represents the different libraries and systems used. There are two elements, Nordic SD - SDK for the Bare Metal system and Zephyr RTOS.

#### Nordic SD/SDK

Nordic Semiconductor provides two libraries to help in the development of applications.

* **SoftDevice (SD)**, Bluetooth low energy (BLE) Central and Peripheral protocol stack
* **Software Development Kit (SDK)**, facilitate firmware development for different devices and applications

different version of the SD and SDK. As the nRF52840 is last SoC release, the version of the SD and SDK used for this project are the most recent.

The SoftDevice used is the S140 and the Software development kit used is nRF5 SDK v13.0.0. A new SDK version was release, v13.1.0, in the middle of the project but not used because the Abstract Layer was already developed.

#### Zephyr RTOS

Zephyr Project needs a Software Development Kit that contains all necessary tools and cross-compilers needed to build the kernel on all supported architectures. The version use for this project was SDK v0.9.1.

Zephyr Project is maintained on a public GitHub repository. Hence, the source code was frequently update to use the last version.

## Abstract Layer

The Abstract Layer interfaces the features of the systems used in the Environment Layer to use the same code for the Driver Layer and for the Application Layer.

The abstract layer is separated in seven parts:

* **iSpi**
* **iI2c**
* **iGpio**
* **iEventQueue**
* **iTimer**
* **iThread**
* **iBleP**, peripheral BLE
* **iBleC**, central BLE

The particularity of a Bare Metal is that it does not use an operating system. Therefore, it has no thread. However, it is possible to use a XF pattern to execute process when an event is push within a queue event. This pattern requires a scheduler to dispatch the event.

The SDK library provides a scheduler but decided to not use it and to create my own scheduler because the use of the SDK scheduler was not convenient to interface the SDK and Zephyr RTOS.

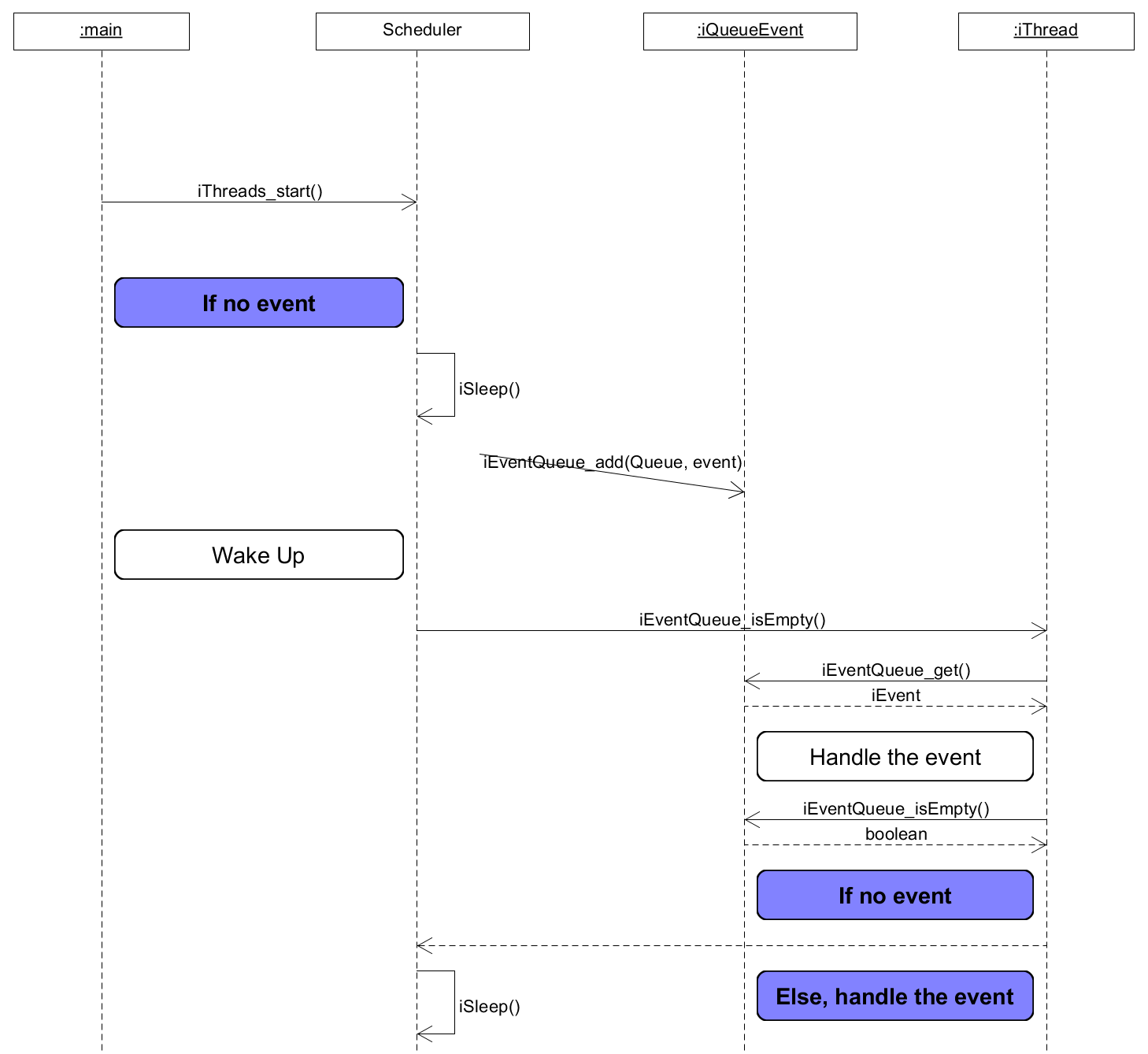


Figure : Scheduler for Bare Metal iThread

To fulfil the condition to use a XF pattern, each iThread must be built as the state machine below:



Figure : iThread architecture

## Driver Layer

The Driver Layer manages the communication with the device of the extension board. It allows to easily configure the device and acquire the data.

There is a driver for each components of the extension board:

* **SWG**, Square Wave Generator, driver of the AD9837
* **ACC**, Accelerometer, driver of the LIS3DH
* **ADC**, A/D Converter, driver of the MAX11200

## Application Peripheral

The peripheral is separated in four **threads**, each thread pop from a different event queue:

* **BLE thread**, disable the drivers when the central is disconnected
* **ACC thread**, notify the central of the axis values and
* **ADC thread**, notify the central of the values
* **SWG thread**, increase the interrupt frequency

Then, various elements pushed events within the event queue:

* **BLE**, events when device is connected or disconnected
* **ACC GPIO interrupt 1**, event when new samples ready
* **ACC GPIO interrupt 2**, event when clicks
* **ACC Button**, to enable or disable the driver
* **ADC Timer**, event when new value ready, timer adapted to the sample rate of the converter
* **ADC Button**, to enable or disable the driver
* **SWG Button**, event to increase the interrupt frequency
* **SWG Button**, to enable or disable the driver

Another part of the application is used for the measurement:

* **GPIO Interrupt**, to measure the hardware interrupt latency
* **Timer**, to measure the software interrupt latency

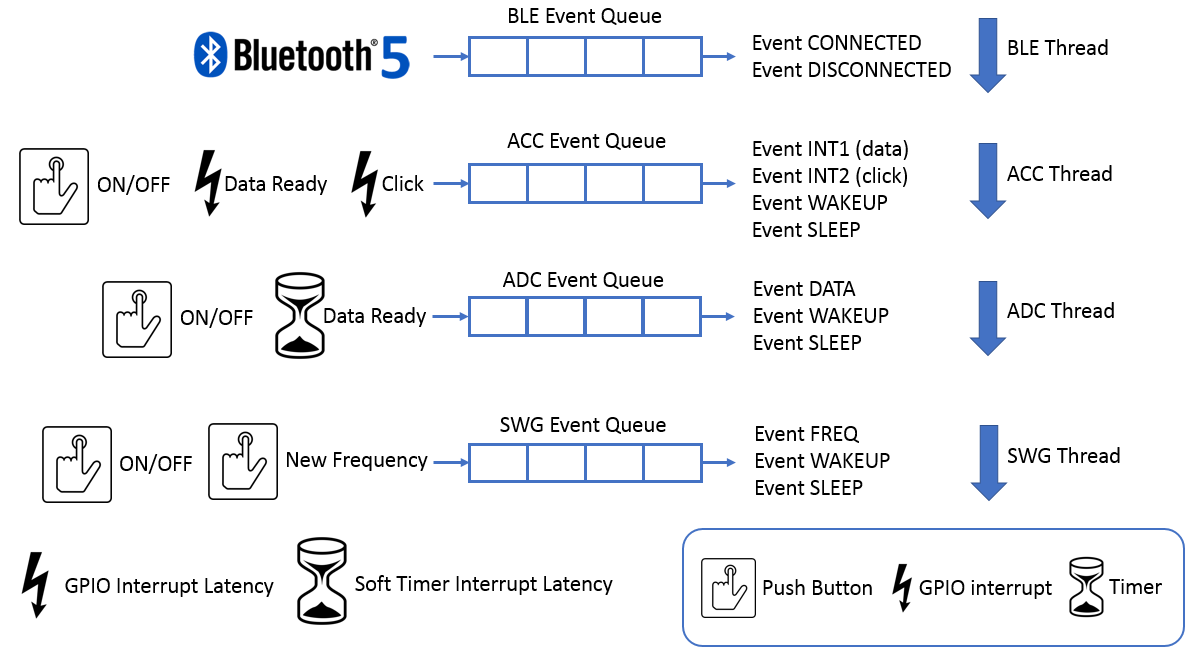


Figure : Peripheral application architecture

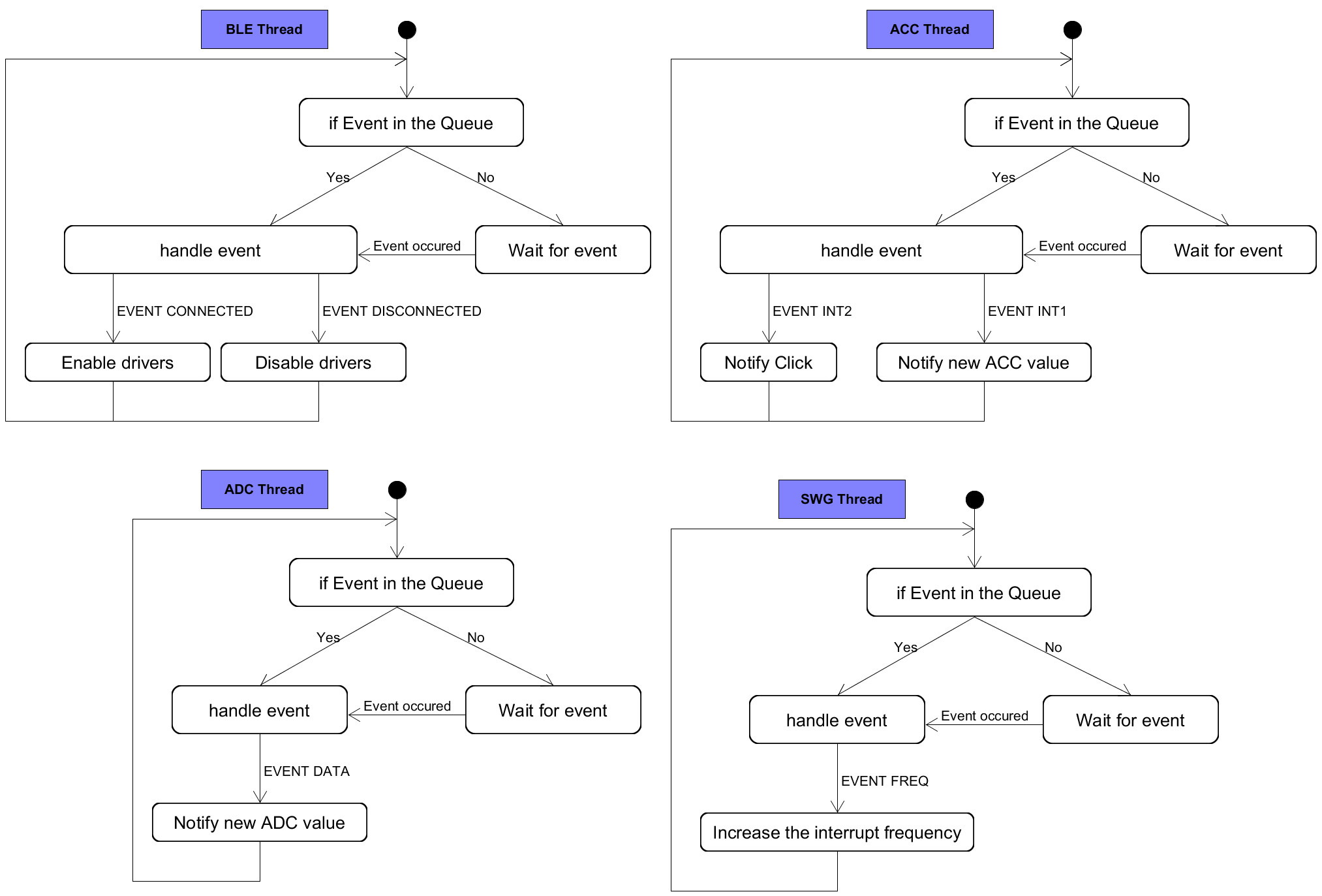


Figure : Threads process of the peripheral application

No BLE services already exist to send data from accelerometer or A/D converter. Therefore, simple custom services are made.

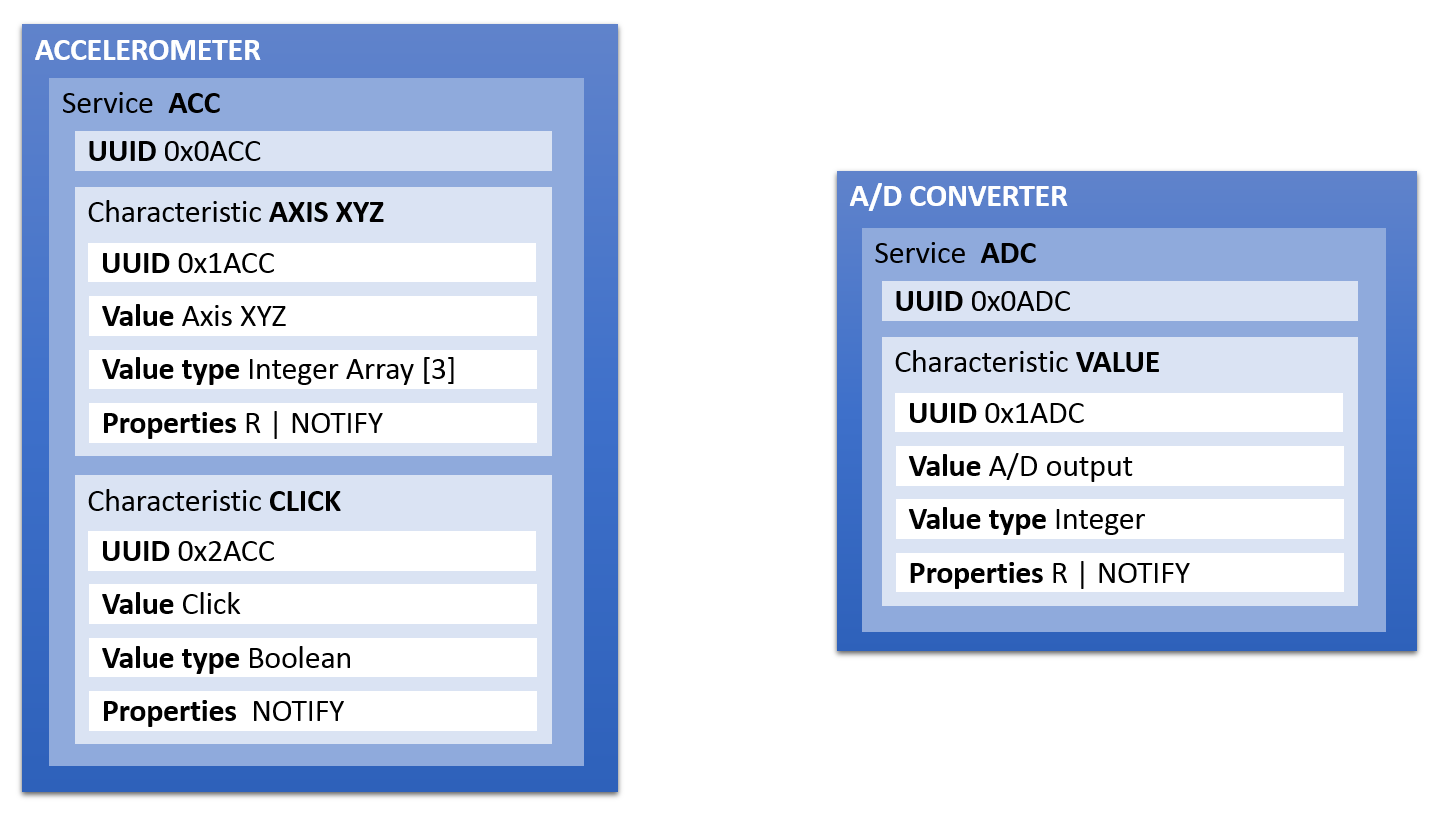


Figure : BLE services for peripheral

### Configuration

It is possible to configure the peripheral to only use some drivers or disable the Bluetooth. The configuration allows to:

* **Enable/Disable** the Bluetooth
* **Set** the Advertisement Interval
* **Enable/Disable** the ACC
* **Set** the data rate of the ACC
* **Enable/Disable** the ADC
* **Set** the data rate of the ADC
* **Enable/Disable** the SWG
* **Set** the interval between each interrupt frequency
* **Enable/Disable** the software timer
* **Set** the interval of the software timer

## Application Central

To test the central, several peripherals communicate with it. However, it is not relevant to use the extension boards created with all the peripherals. More, the nRF52840 Development Kit is in limited quantity. Hence, an application peripheral is developed on nRF52832 Development Kit to emulate values.

### E-Peripheral

The application is called E-Peripheral for Emulated Peripheral. The E-Peripheral is separated in three **threads**, each thread pop from a different event queue:

* **BLE thread**, disable the emulated drivers when the central is disconnected
* **ACC thread**, notify the central with emulated values
* **ADC thread**, notify the central with emulated values

Then, various elements pushed events within the event queue:

* **BLE**, events when device is connected or disconnected
* **ACC Timer Data**, event to emulate new samples ready
* **ACC Timer Click**, event to emulate clicks
* **ACC Button**, to enable or disable the emulated driver
* **ADC Timer**, event to emulate new values ready,
* **ADC Button**, to enable or disable the emulated driver

To indicate the state of the peripheral, two LEDs are used:

* **LED Started**, which indicates when the application is running
* **LED Connected**, which indicates when the device is connected to the central

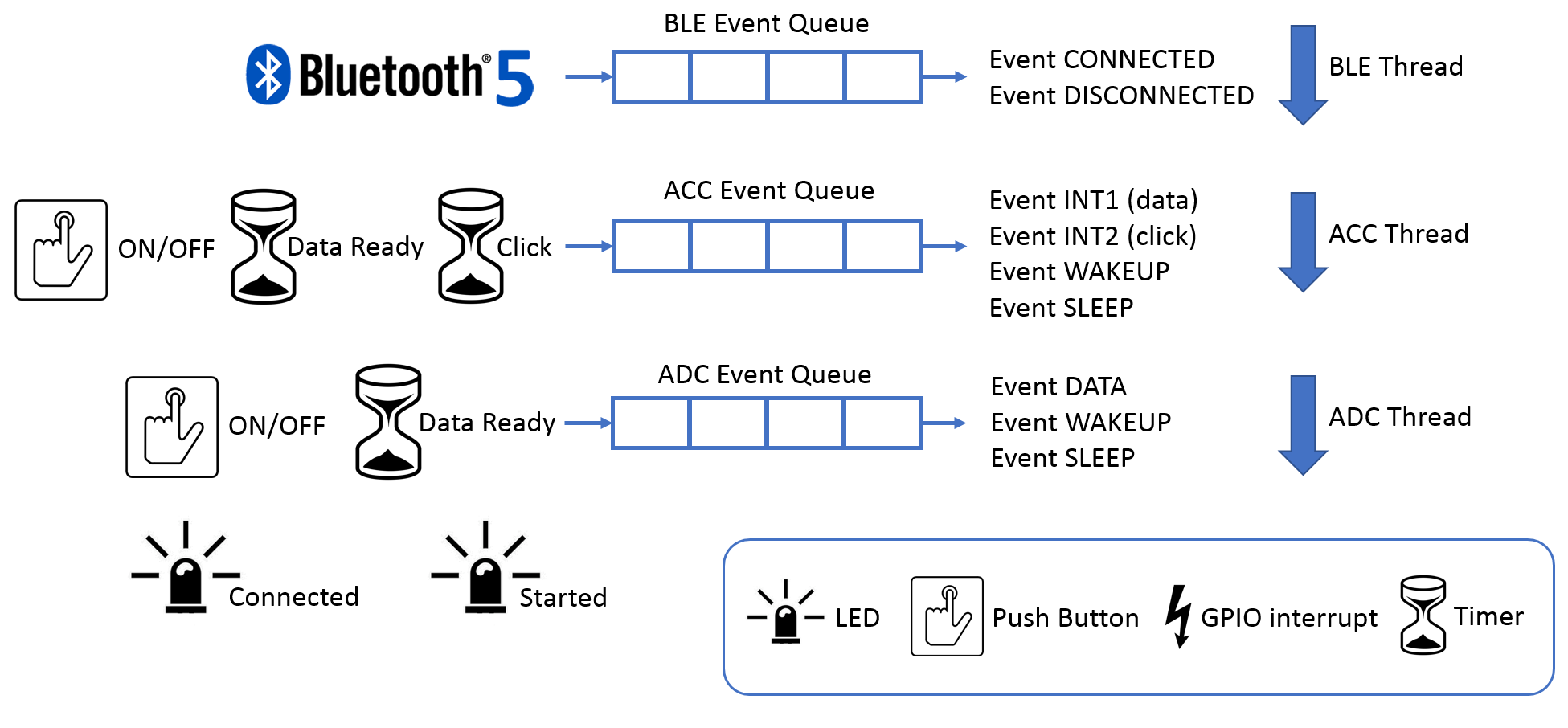


Figure 24: E-Peripheral application architecture

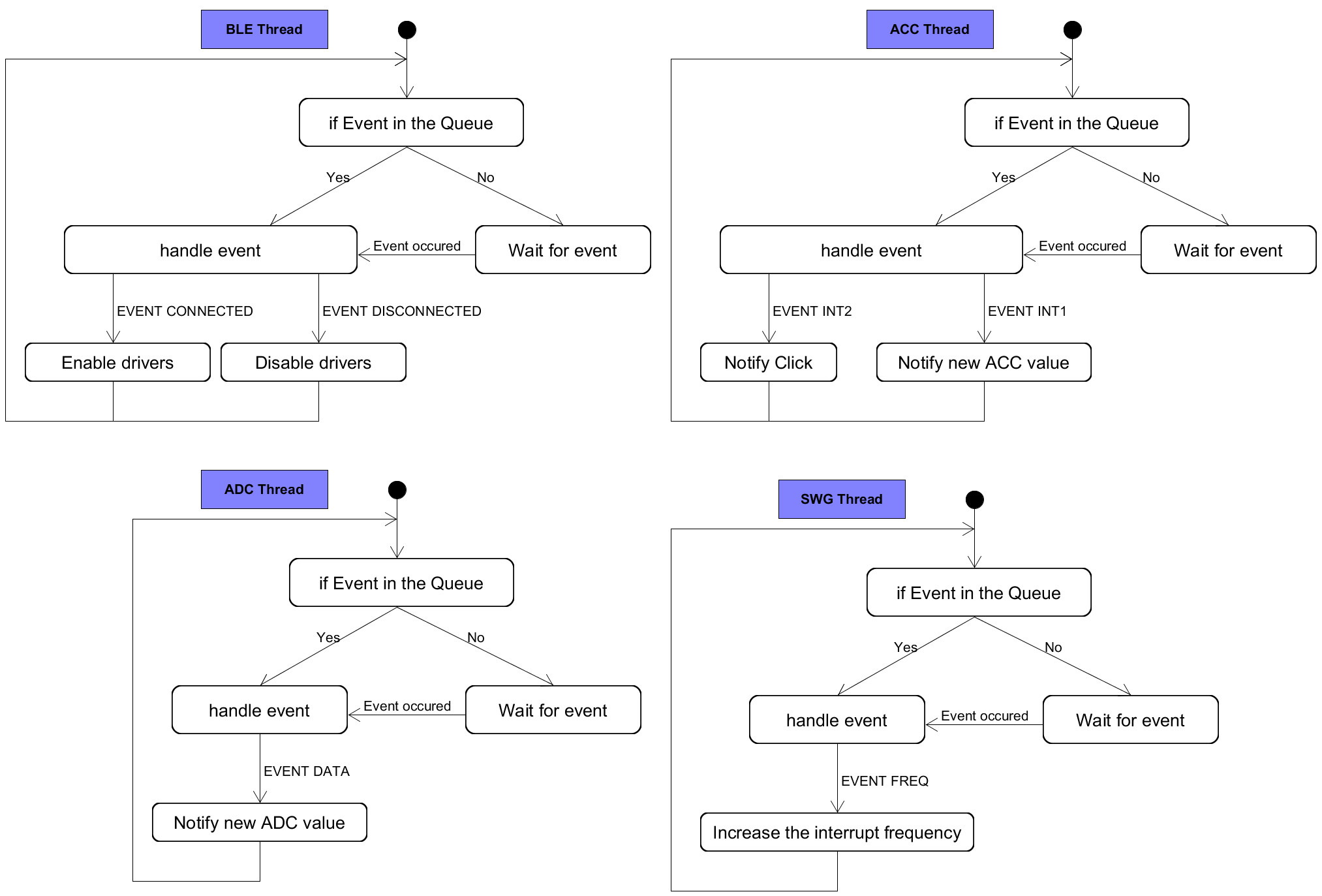


Figure 25: Threads process of the E-Peripheral application

The BLE services used are the same than the application peripheral developed for the measurements.

#### Configuration

The configuration of the E-Peripheral allows to:

* **Set** the Advertisement Interval
* **Set** the data rate of the emulated ACC
* **Set** the data rate of the emulated ADC

### Central

The Central will search all the E-Peripherals by them name. Then, the central will search the services of the central, ACC Data, ACC Click, ADC Data. The Central is separated in two **threads**, each thread pop from a different event queue:

* **BLE thread**, which enables the notification of the connected peripheral
* **SWG thread**, which increases the interrupt frequency

Then, various elements pushed events within the event queue:

* **BLE**, events when device is connected, disconnected or the services discovery is finished
* **SWG Button**, event to increase the interrupt frequency
* **SWG Button**, to enable or disable the driver

Another part of the application is used for the measurement:

* **GPIO Interrupt**, to measure the hardware interrupt latency
* **Timer**, to measure the software interrupt latency

The Last part of the application takes care of calling the handler for notification or indication from a peripheral:

* **ACC Data**
* **ACC Click**
* **ADC Data**

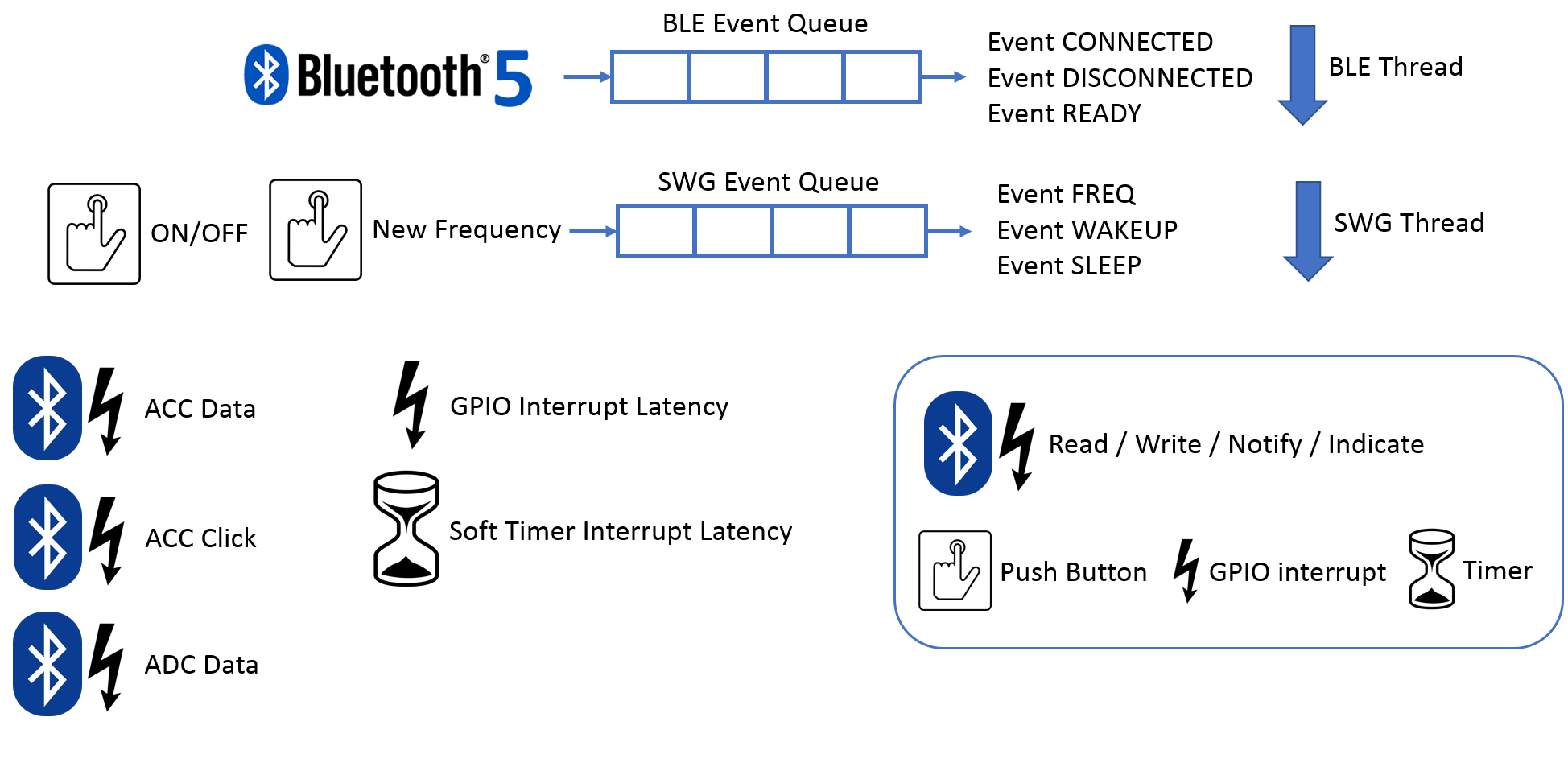


Figure 24: Central application architecture

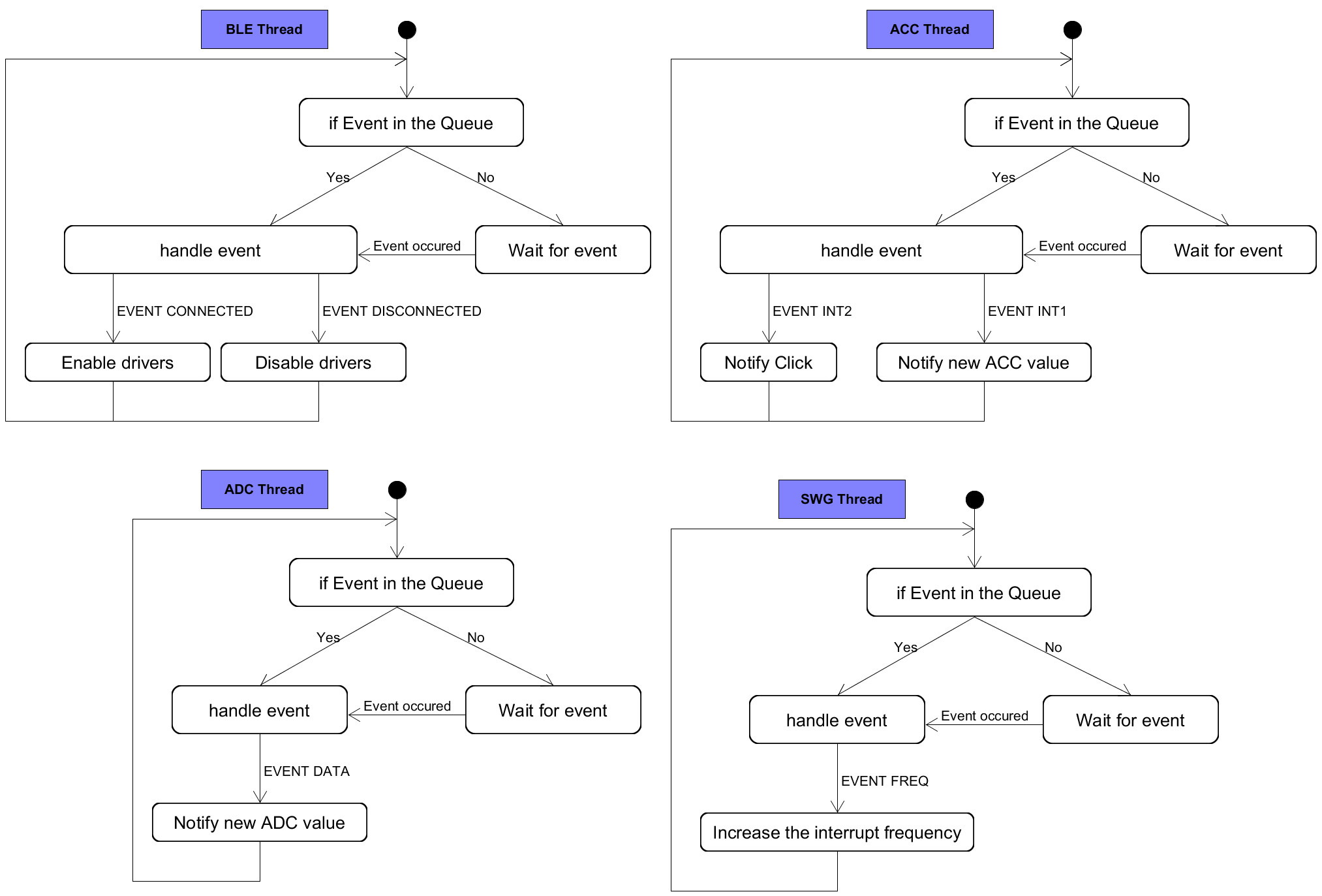


Figure 25: Threads process of the Central application

#### Configuration

It is possible to configure the peripheral to only use some drivers or disable the Bluetooth. The configuration allows to:

* **Enable/Disable** the Bluetooth
* **Set** the Scanning Interval
* **Set** the connection interval
* **Enable/Disable** the SWG
* **Set** the interval between each interrupt frequency
* **Enable/Disable** the software timer
* **Set** the interval of the software timer

## Annexes

## Sources

<https://infocenter.nordicsemi.com/index.jsp>

<https://nexus.zephyrproject.org/content/sites/site/org.zephyrproject.zephyr/dev/api/api.html>

<https://github.com/zephyrproject-rtos/zephyr>

# Results

## Equipment

## Error

## Results

### Peripheral

### Central

## Comparisons

### Peripheral

### Central

## Annexes

# Conclusion

# Abbreviation

**SoC**: System on Chip

**SD**: SoftDevice

**SDK**: Software Development Kit

**BLE:** Bluetooth Low Energy

GAP

GATT

ATT

SMP

L2CAP

HCI

LL

PHY

MTU

ACC

ADC

SWG

PPI  
GPIOTE