

---

# RTLS Toolbox

The RTLS (Real Time Localization System) Toolbox, is a collection of RTLS techniques that can be implemented on TI's standard Bluetooth Low Energy radios in the CC26xx series. These techniques provide raw data that can be utilized for developing localization algorithms and secure range bounding other Bluetooth Low Energy nodes. The two main techniques included in the RTLS toolbox are RSSI and [Bluetooth Core Specifications Version 5.2 Angle of Arrival](#).

RSSI details the Received Signal Strength Indication of an incoming signal and is commonly leveraged for deriving the distance between a receiver and a transmitter through the process of trilateration in localization algorithms. The Bluetooth Low Energy stack enables developers to receive the RSSI of an incoming Bluetooth packet.

[Bluetooth Core Specifications Version 5.2 Angle of Arrival](#) is a technique for finding the direction that an incoming Bluetooth packet is coming from, creating a basis for triangulation. The device samples an incoming constant tone and as I/Q data. This raw I/Q data represents the amplitude and phase data of a signal and this data can be used to derive the angle the device transmitting the constant tone.

For detailed information on the specific example, see the relevant README.html file in the `simplelink_cc13x2_26x2_sdk_x_xx_xx_xx/examples/rtos/CC26X2R1_LAUNCHXL/ble5stack/PROJECT` folder.

Using the raw data provided by the RTLS Toolbox, TI is enabling developers to improve localization algorithms based on Bluetooth technology by delivering more data that can be leveraged for trilateration and triangulation.

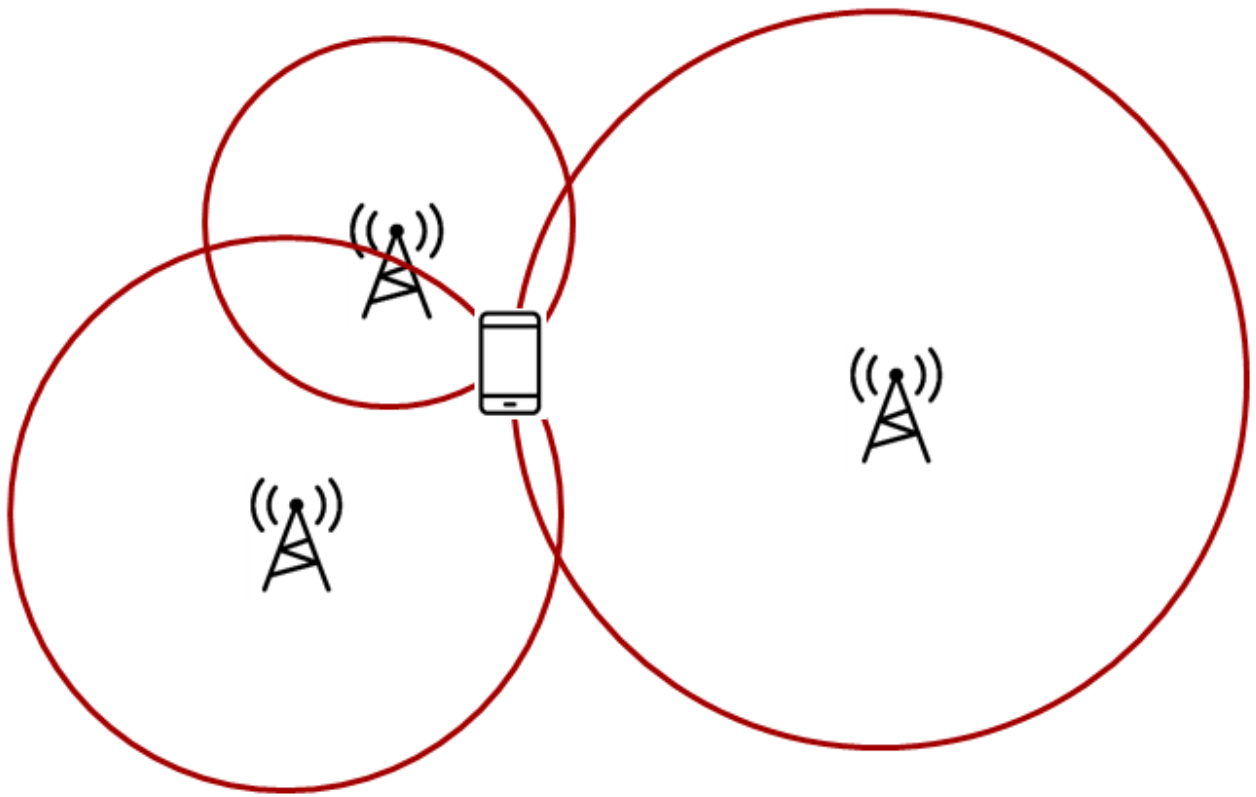
The inherent flexibility of the CC13x2 or CC26x2 RF Core is what enables this significant extension of functionality. The main advantages using the CC13x2 or CC26x2 are that customers can start adding RTLS features and security with little extra cost, very little extra energy consumption and no increase in peak power.

There are two fundamentally different approaches for localization:

---

**Trilateration**

---



Trilateration is where you know the distance between a reference node and a target node.

[RSSI Based Localization](#) gives you the distance from the receiver to the transmitter.

#### ! Note

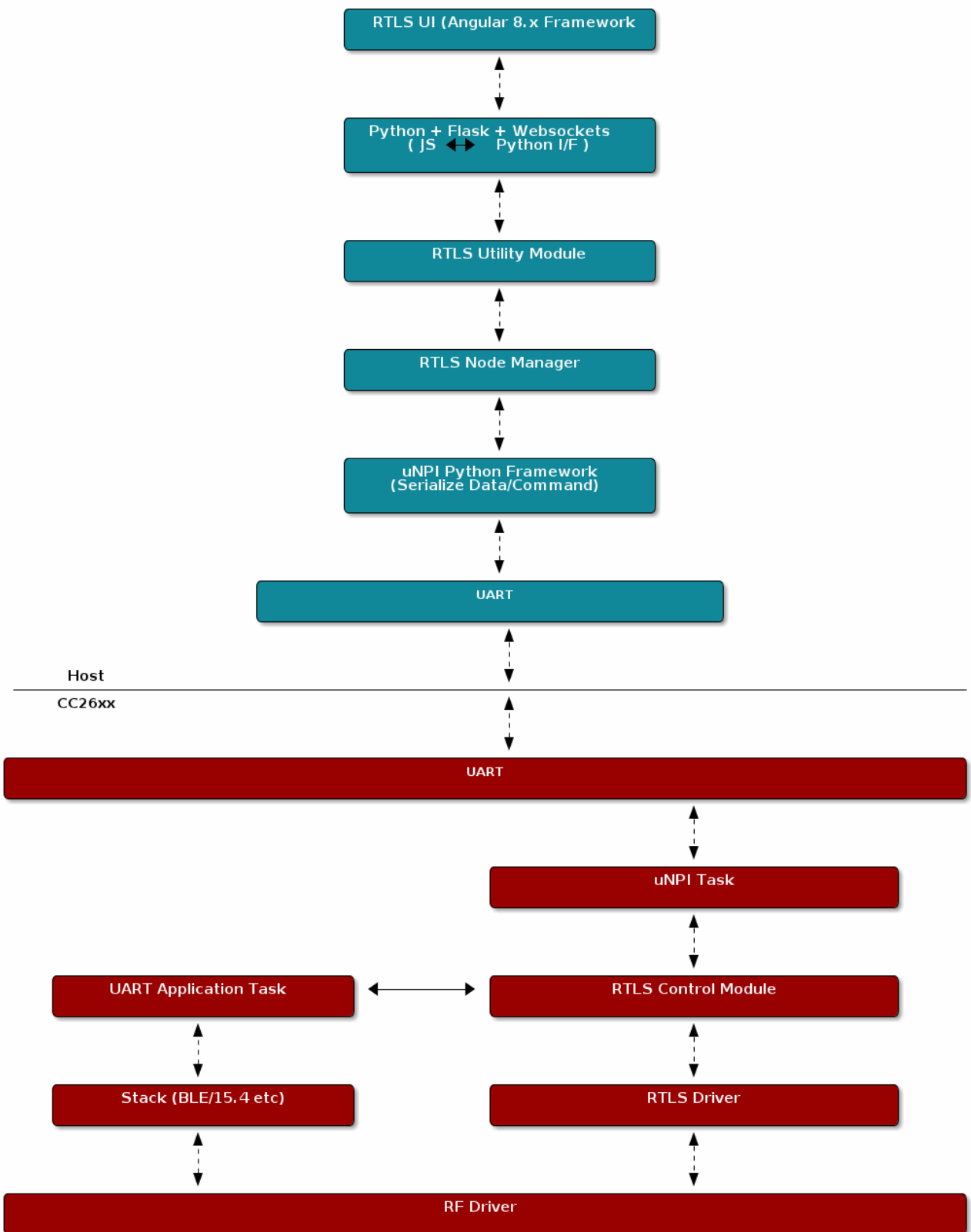
Existing examples with no external control interface are discontinued. All RTLS examples now have an external control interface.

## General RTLS Software Architecture

The diagram below shows the RTLS software architecture.

With multiple nodes that collect localization data, it is important to have an architecture that supports control of these nodes. It must be possible to configure and trigger localization and it must be possible to retrieve localization data. We achieve this by reuse of our Network Processor Interface (NPI). It is basically a means to support Remote Procedure Calls (RPC) over a serial interface. Note that the architecture is such that it is possible to replace NPI with your own preferred serial protocol.

In our example we use UART as the serial transport layer. This is because it is readily available on host PC as UART over USB, and our LaunchPads include a UART to USB bridge. Along with the embedded examples we provide PC software to act as host, to control, retrieve and present localization data. This also allows much quicker customer performance characterization, as well as configuration of important parameters, not to mention the ability to develop and test higher level post-processing algorithms.



## RTLS Toolbox

The RTLS toolbox is a collection of software that is purposed for the localization use case. A table below summarizes each software component in the toolbox as well as its applications for localization. The software components below will run on the embedded nodes.

Software Component	Usage
--------------------	-------

Software Component	Usage
BLE-Stack	Advertising, scanning, connection, exchanging RTL
Connection Monitor	Follows a BLE connection using Micro BLE-Stack
Angle of Arrival	Radio patches and driver to read AoA data embed
Unified Network Processor Interface (UNPI)	A serial protocol including packet format and hanc
RTLS Control Module	Implements the command and event set used to c

The software components in the following table run on a PC. TI has setup a PC based environment to facilitate in evaluating and prototyping various RTLS algorithms. Once the algorithms are complete, the various PC components above can be migrated to an embedded device.

Software Component	Usage
RTLS Util	Event handling and distribution across nodes. Implements results queu
RTLS Node Manager	Framework for sending and receiving RTLS commands and events from
Websocket Server	Implements the server side of a TCP socket. Intended to bridge serial cc
RTLS UI GUI	A GUI for aggregating and logging RTLS communication and graphing A

## RTLS Utility

The RTLSUtil module is a convenience layer that is built on top of the node manager that implements event handling and blocking as required by the RTLS network (e.g. seed distribution). Additionally the RTLSUtil module is able to setup worker threads for data processing of results from the device as well as creating queues to store the incoming data.

## RTLS Node Manager

The RTLS Node Manager provides the following functionality:

- A bridge between RTLS US GUI and CC13x2 or CC26x2
- Holds the state of each device and takes care of configuration
- Enables the simplicity of RTLS Control Module and RTLS UI GUI
- Minimizes amount of the transactions between RTLS UI GUI and the rest of the system

Due to the above functionality, the Node Manager can operate on user's CPU and help with slow bus (LIN/CAN).

# RTLS Control Module

The RTLS Control Module is an on-chip module which runs as a RTOS Task and it provides the following functionality:

- Parsing commands coming from Node Manager
- RTLS Driver configuration and operation
- RF and NPI message queue

## RTLS Roles and Topology

Each node in an RTLS network utilizes the software components listed above in a different way to perform a specific task related to localization. These capabilities map to a role within the RTLS network and ultimately are implemented by sample applications within the SDK. There are three examples: `rtls_master`, `rtls_slave`, and `rtls_passive`. The capabilities of these examples are explained below. All embedded nodes implement UNPI and act as slaves in the UNPI protocol. Additionally all embedded nodes have the RTLS Control module implemented for processing RTLS commands from the UNPI interface.

### Note

The following subsections aim to describe what localization roles are implemented by the sample applications in the SDK. This is not a comprehensive list of what is possible on each node, but rather an explanation of what the examples will do out of the box. For a list of potential combinations, please see Role Combinations.

## Master

The RTLS master runs a full BLE-Stack and acts as a central device. It will scan and connect to the RTLS slave over BLE. Once a connection is established the RTLS Master will do the following:

- Share the connection parameters (access address, master sleep clock accuracy, and CRC init) with the PC.
- Use the BLE link to share AoA parameters with the peripheral device.
- Implements the AoA master role
- Does not send out AoA packets, but configures the slave to do so (in the case of connection AoA)
- Synchronizes with periodic advertisements sent by the slave (in the case of connectionless AoA)
- Receives packets with CTE and performs in-phase and quadrature component (IQ) sampling

## Slave

The RTLS slave runs a full BLE-Stack and acts as a peripheral device. This is the device that is to be located. In the case of connected AoA, the slave device will advertise and enter a connection with the RTLS Master. In the case of connectionless AoA, the slave sends out periodic advertisements the master can synchronize with.

- Sends data packets with AoA tone embedded using Constant Tone Extension (CTE)
- Advertises special string to be detected by `rtls_master` (in the case of connection AoA)
- Advertises periodic advertisements (in the case of connectionless AoA)
- Implements AoA slave role
- BLE-Stack peripheral role
- Wireless/battery operated, not connected to PC

## Passive

The RTLS passive does not actively participate in the BLE connection between the RTLS master and slave. Instead, it uses the [Micro BLE Stack](#) in connection monitoring mode to follow the connection. To do this, the passive device relies on the Master to distribute the connection parameters once a connection is formed. The passive node does the following:

- Implement AoA passive role
- Receives packets with CTE and performs in-phase and quadrature component (IQ) sampling
- Implements the Micro BLE-Stack connection monitoring application layer. See [Connection Monitor \(CM\) Application](#) for more information on the connection monitor.

### ! Note

The `rtls_passive` device is not used for Connectionless AoA.

## PC/Central Processing Node

The PC node is responsible for controlling the embedded RTLS nodes by sending commands and processing events. In the SDK, this is realized by a combination of a Python layer that implements the UNPI master role and a websocket server that translates UNPI commands to a socket interface that is used by the GUI Composer application running in the browser.

This software is intended to use as a framework for extracting data from the embedded nodes and using it to prototype high level RTLS algorithms on the PC.

### ! Note

In a final product, these algorithms may be implemented on an embedded device or even perhaps the RTLS master node. TI has provided PC software to aid in data plotting and prototyping various algorithms.

The PC implements the following roles in Python:

- UNPI master
- COM port interface
- Implementation of RTLS UNPI subsystem/command set
- Websocket server

The PC implements the following roles in RTLS UI GUI/JavaScript:

- Websocket client on localhost
- Graphing and logging data from websocket
- Parsing JSON objects to extract RTLS data
- Issue commands to RTLS nodes via websocket to UNPI conversion
- Enumerate devices
- Distribute connection parameters to passives

## Role Combinations

### Note

In all of the tables following (BLE, AoA) features listed as optional are **not implemented** by the sample applications included in the SDK, but are valid and possible configurations that can be implemented by the user.

For example: `rtls_master` could be a multi-role device or `rtls_passive` could operate as AoA master.

## BLE-Stack

The Bluetooth Core Specifications Version 5.2 allows for devices to operate in various roles as well as combinations of roles. The table below shows the required and optional features for each example.

Example Name	Central	Peripheral	Broadcaster	Observer
RTLS Master	R	O	O	R
RTLS Slave	O	R	R	O
RTLS Passive	No	No	O	R*

Legend:

- R: Required
- O: Optional
- No: Not supported

### ! Note

The R\* above denotes that while the connection monitor is capable of scanning for beacons, it is also required that the connection monitor follow a connection. The monitoring role is not officially defined by the Bluetooth Spec, but it is a critical functionality in the `rtls_passive`.

## AoA

The Bluetooth Core Specifications Version 5.2 defines multiple roles for both connected and connection-less AoA. The following configurations are supported by the examples in the [SimpleLink CC13x2 / 26x2 SDK](#).

Example Name	Send CTE	Perform IQ Sampling
RTLS Master	No	R
RTLS Slave	R	No
RTLS Passive	N/A	O

Legend:

- R: Required
- O: Optional
- No: Not supported
- N/A: Not applicable

For connection AoA, an AoA Master / Passive is capable of connecting to / monitoring up to and including eight AoA slaves simultaneously.

For connectionless AoA, an AoA Master is capable to synchronize with up to 40 slaves. When the number of slave synchronized with the master is big, some of the periodic advertisements may not be received. In addition, the number of buffers allocated to CTE sampling has to be big enough (see `MAX_NUM_CTE_BUFS`)

## RTLS Driver

The purpose of RTLS Driver is to handle the Direct Call implementation of the RTLS module.

For more detail please take a look at [AoA Driver](#) section.



# Physical Considerations

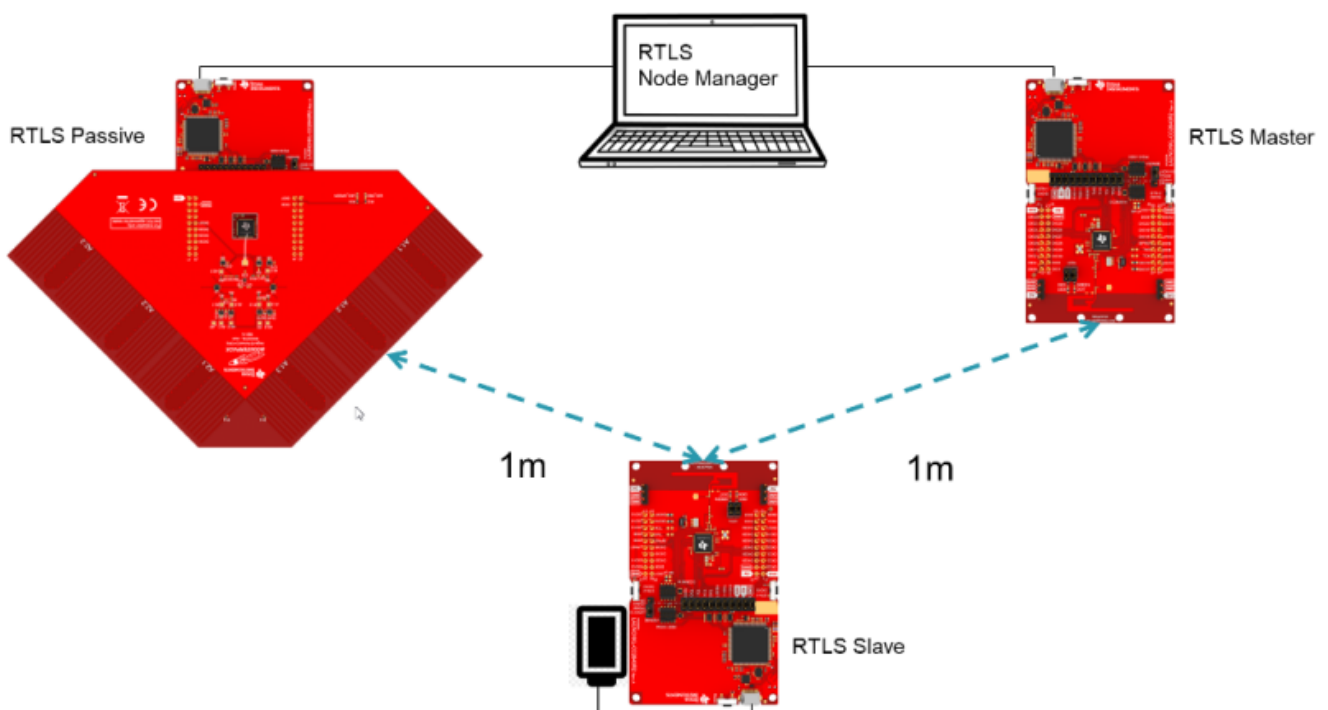
Before evaluating the RTLS solution, it is important to consider the environment. All radio communication protocols can be susceptible to [multi-path fading](#), and RTLS based systems are not exempt. It is important to control the environment when evaluating or at least be aware of the potential effects of multi-path on the results.

It is recommended to evaluate the RTLS solution in an area that optimizes RF conditions. This includes:

- An open space with no large metal or concrete obstructions (pillars, poles, etc.)
- Relatively few interference sources (i.e. Wi-Fi access points, etc.)
- Raised platforms for the nodes made out of cardboard ~1 m off the ground.

A desk environment generally has sub-optimal RF conditions and should be avoided.

See the image below for a recommended layout of the nodes during evaluation, this is a 2D image where all devices are laying on a flat surface “pointing” as shown in the picture. In this case each node should be placed on a box so that it does not sit directly on the ground.



## ⓘ Note

The `rtls_master` device can be set to sample the CTEs too - the default example enables this possibility. In that case, an antenna board needs to be set on the `rtls_master` device and the `rtls_passive` can be omitted.

For angle of arrival application, we have created [Bluetooth Angle of Arrival Antenna Design](#) to further explain what users should look for when making their own AoA board.

## Test mode

Test mode is used for Bluetooth certification. RTLS test mode can be enabled in the *host\_test* example by adding the pre-defined symbols *USE\_RTLS*, *RTLS\_CTE*, and *RTLS\_CTE\_TEST*. For further description of the above preprocessor defines, please see [Table 18.](#)

## RSSI Based Localization

RSSI details the Received Signal Strength Indication of an incoming signal and is the most commonly used method of trilateration localization in Bluetooth. TI Bluetooth low energy stack enables developers to receive the RSSI of an incoming Bluetooth packet with can be used to enable RTLS algorithms.

RSSI localization is based on the the [Frii's Transmission Equation](#). The core concept is that received signal strength is proportional to the distance of the transmitting node. The graphic below describes how RSSI can be used to estimate distance.

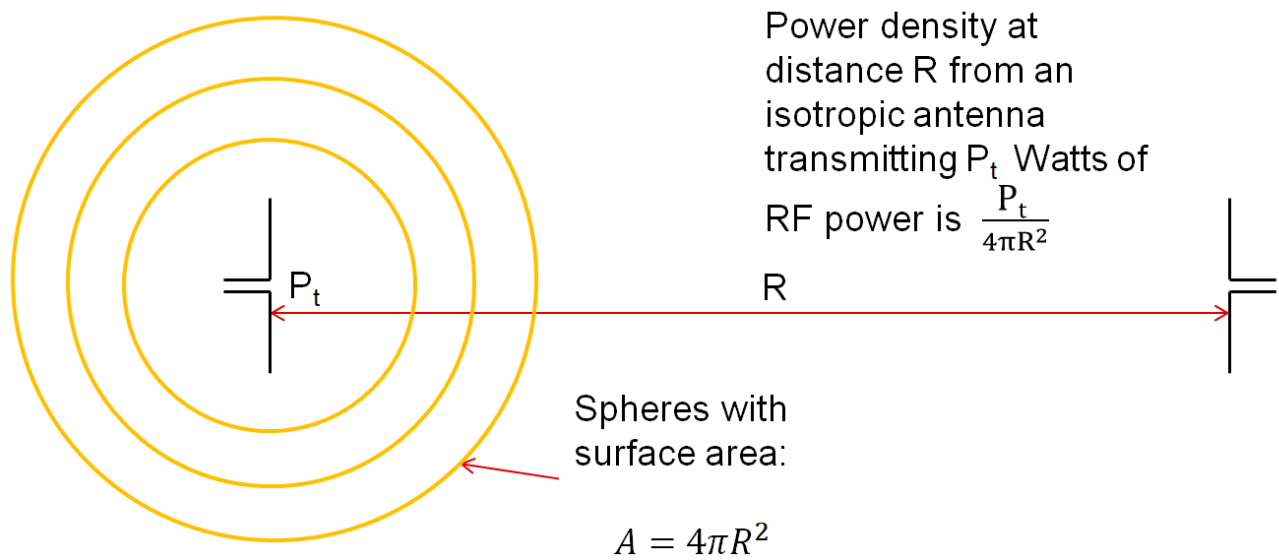


Figure 121. Frii's Equation Relationship between received power and distance.

While RSSI based localization is the most commonly used method in today's RTLS systems, it also faces challenges that need to be overcome:

- Accuracy can be influenced by the presence of reflections and obstructions
- No Relay Attack protection

Some of these challenges with RSSI can be overcome through smart system design. TI's RTLS Toolbox enables developers to monitor the connection between a master and slave and get independent RSSI measurements from the same packet via the Connection Monitor. This approach gives more data and reference points that can be used to help improve the resolution of an RTLS application.

For future RTLS applications, it is recommended to combine RSSI with the other localization techniques such as AoA to help improve the accuracy and security.

## Reading RSSI

The [Micro BLE Stack](#). (foundation of the [Connection Monitor \(CM\) Application](#)) and the full BLE5-Stack provide APIs to read RSSI information of the received packet. The following sections will describe how to extract RSSI information from the received packet.

## Connection Monitor

The connection monitor will keep an array of connection information for each connection it is tracking. This includes master and slave RSSI from the last scan. These fields can be found in the `ubCM_ConnInfo_t` structure. The fields are `rssiMaster` and `rssiSlave` respectively.

RSSI information is valid after the monitor complete callback is invoked (when the monitoring scan is complete for a connection event).

The `rtls_passive` sample application shows an example of extracting slave RSSI in `RTLSPassive_monitorCompleteEvt`.

## Keyless entry with TI connection monitor

This [video](#) explains the concept of using TI Connection Monitor to monitor Bluetooth Low Energy communications to estimate the distance between the connection monitor and Bluetooth LE enabled devices. This can be used in applications such as car access or remote keyless entry.

## BLE(5)-Stack

When using the BLE5-Stack, the `Gap_RegisterConnEventCb` ([Generic Access Profile \(GAP\)](#)) will provide RSSI from the last connection event. It can be used in the central or peripheral configuration. The connection event callback is already the synchronization method used by the `RTLSCtrl` module for reporting data to the PC/Node Manager. See [Gap\\_ConnEventRpt\\_t](#) for information on how to extract the RSSI from connection event callbacks.

In the case of communication that consists of multiple packets, only the RSSI of the last received packet is reported. There is no averaging over channels. In general, the received signal strength will vary across channels due to varying channel conditions and antenna gains. This has the following consequences in regards to BLE:

- Advertising: The RSSI will only be reported on the last advertisement channel that data is received on.

- Connection Events with multiple data packets: These occur on the same channel but the RSSI will still only be reported for the last data packet that was received.

### **Note**

The algorithm that calculates RSSI is the same algorithm for all BLE channels.

## **Calculation Detail**

The RSSI value is a measurement of the actual power level delivered to the input of the reference design RF matching and balun. A valid range for RSSI signals is approximately -90dbm to -30dBm. Note that it is not easy to directly correlate RSSI to distance as it is highly dependent on the antenna design and polarization of both the transmitter and the receiver antenna.

The RSSI can have negative tolerance which means that if you are receiving on the sensitivity limit, reported RSSI can potentially be lower than this. The sensitivity can be found in the device datasheet and is in accordance with Bluetooth requirements.

The RSSI performance will be the same between both single ended and differential configurations and the expected accuracy is +/-4 dbm as stated in the datasheet.

## **Advantages of a connection based RSSI system**

A connection based RSSI system relies on a Bluetooth Low Energy connection. The RSSI of the signal received during the connection event is measured. The advantages of this technique against connectionless based RSSI system are:

1. The Bluetooth Low Energy connection ensures the identity of the target device.
2. The number of possible RF channels used is increased from 3 to 37. This increases the system's robustness to interferences.
3. The energy consumption of the passive devices is smaller as they are not continuously scanning.

## **Advantages of a connectionless based RSSI system**

A connectionless based RSSI system locates beacons without forming a Bluetooth Low Energy connection. The RSSI of the advertisements is measured. The advantages of this technique against connection based RSSI system are:

1. The complexity of the system is reduced as there is no need for sharing the connection parameters between the passive-nodes.
2. The number of targets that can be tracked simultaneously is slightly bigger (fewer resources are required to keep track of a non-connected device than a connected device).

# Angle of Arrival

---

Angle-of-Arrival (AoA) is a technique for finding the direction that an incoming Bluetooth packet is coming from, creating a basis for triangulation.

An array of antennas with well-defined properties is used, and the receiver will switch quickly between the individual antennas while measuring the phase shift resulting from the small differences in path length to the different antenna.

These path length differences will depend on the direction of the incoming RF waves relative to the antennas in the array. In order to facilitate the phase measurement, the packet must contain a section of constant tone (CT) where there are no phase shifts caused by modulation.

## Packet Format

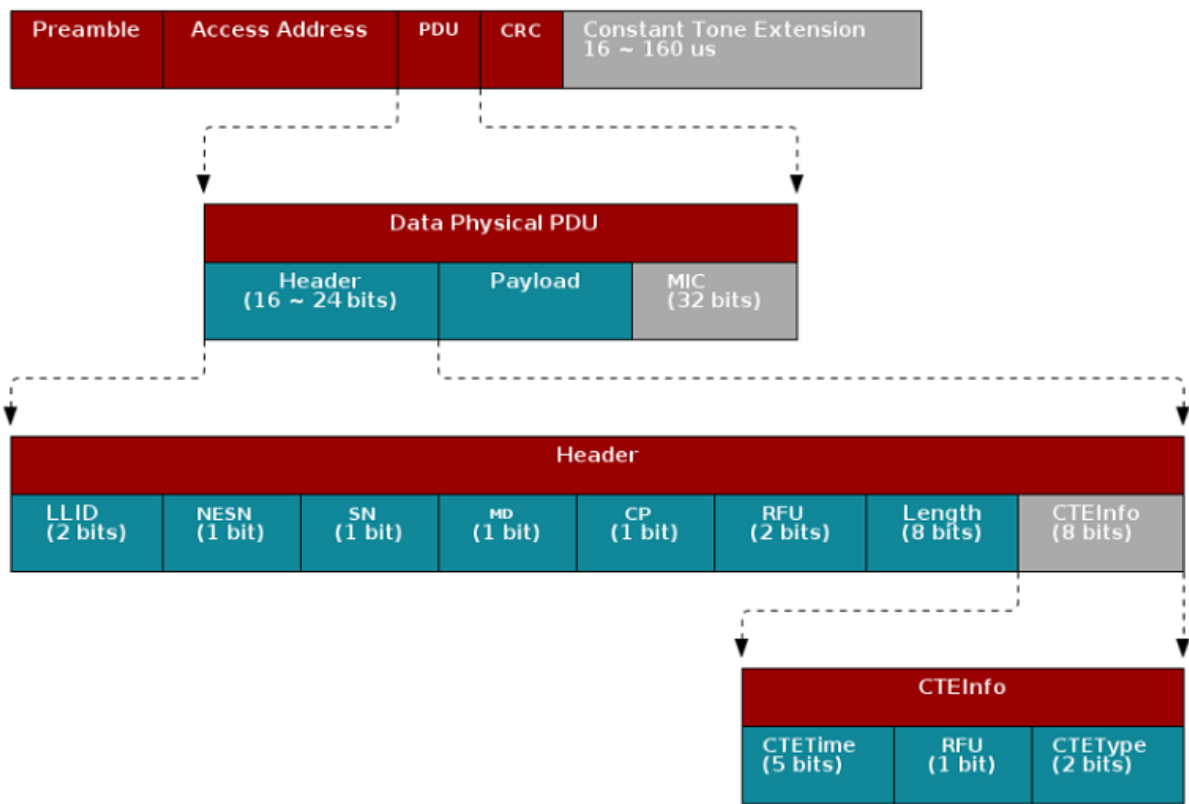
In order to get a good estimate of  $\phi$  (phase), all other intentional phase shifts in the signal should be removed.

Bluetooth Core Specifications Version 5.2 introduces AoA/AoD which are covered under [Direction Finding Using Bluetooth Low Energy Device](#) section and it also specifies the following states can support sending direction finding packets:

1. Periodic advertising; also called [Connectionless CTE](#)
2. Connection; also called [Connection CTE](#)

The theory behind AoA/AoD and Connectionless/Connection CTE(Constant Tone Extension) is the same, therefore, we will only focus on Connection CTE AoA. Both Connection CTE and Connectionless CTE are provided in BLE5-Stack.

First let's take a look at the payload. By adding a section of consecutive 1's at the end of connection packet, effectively transmitting a single tone at the carrier frequency + 250 kHz.



In the header, the CP bit ( `CTE Present` ) determines whether header contains `CTEInfo` or not.

The `CTEType` field of `CTEInfo` further specifies which type of direction finding packet this is and the `CTETime` field specifies the duration of the CTE.

Table 21. CTEType value

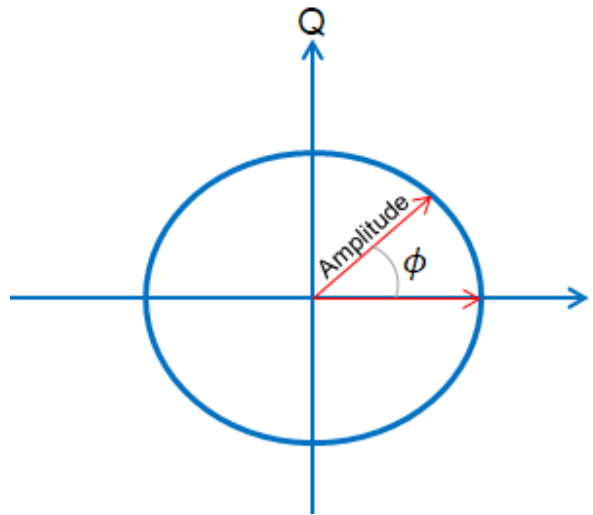
Value	Description
0	AoA CTE packet.
1	AoD CTE with 1 $\mu$ s slots.
2	AoD CTE with 2 $\mu$ s slots.

The value of `CTETime` should be within 2~20 and it's interpreted as in 8us unit. That means when `CTETime` is set to 20, there will be CTE at the end of connection packet which lasts  $8*20 = 160(us)$ .

This gives the receiver time to synchronize the demodulator first, and then store I and Q samples from the single tone 250 kHz section at the end into a buffer and the buffer can then be post-processed by an AoA application

! Note

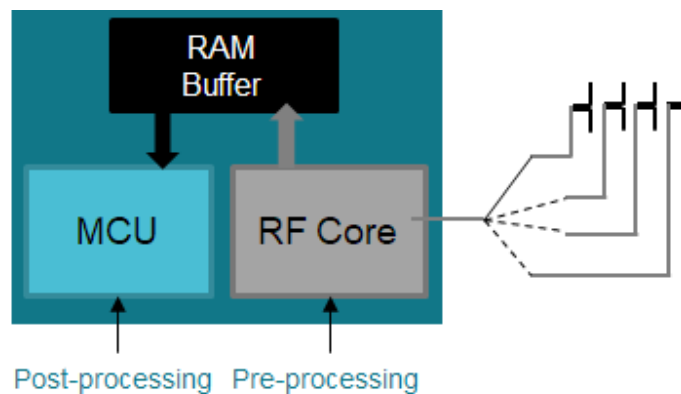
The I/Q Data Sample is the coordinates of your signal as seen down the time axis. In fact, I/Q data is merely a translation of amplitude and phase data from a polar coordinate system to a Cartesian (X,Y) coordinate system and using trigonometry, you can convert the polar coordinate sine wave information into Cartesian I/Q sine wave data.



## Integration

The I and Q samples from the transmitted carrier frequency + 250 kHz tone can be captured, pre-processed, and buffered by the RF Core without any load on the main MCU.

Due to the pre-processing, the application can determine the phase shift without having to remove DC offset or IF first, significantly simplifying the estimation process and leaving the application MCU free to do more on top.



For `rtls_passive`, the I/Q sampling rate is currently not configurable and it is 4 MHz. Each I/Q pair occupies 32 bits space in radio RAM and the radio RAM can store up to 512 samples (i.e. 2048 bytes - 2 kB). This limitation is only due to the micro-stack which limits the amount of data read in the Radio RAM. The radio RAM length is 4 kB.

With sampling rate 4MHz, there will be 16 I/Q pairs every 4us, which equals to  $16 * 4$  (one I/Q pair takes up 4 bytes space) = 64 bytes per 4us.

That means even if the CTE is 160us long with 4MHz sampling rate, the radio RAM for `rtls_passive` can only store I/Q data for 128us duration. There is currently no workaround for it unless we shorten the CTE length. This is not an issue for `rtls_master`.

### ⓘ Note

For `rtls_passive`, I and Q samples only have 13 bits resolution even though they occupy 16 bits space in RF Core RAM. Since they only have 13 bits resolution, the maximum and minimum value you will observe as signed integers are [4095, -4096].

For `rtls_master`, the I/Q [sampleRate Setting](#) is configurable and it's from 1MHz up to 4MHz. Each I/Q pair occupies 32 bits space in radio RAM and the radio RAM can store up to 624 samples (2496 bytes - around 2.5kB). When the sampling frequency is maximum (4 MHz), exactly 624 samples are stored during each CTE. (The CTE lasts 160 us, minus 4 us of guard period. When sampling frequency is 4 MHz, in 156 us,  $156 * 4 = 624$  samples are stored. More details are provided in [Valid I/Q Samples For Angle Calculation.](#))

### ⓘ Note

For `rtls_master`, I and Q samples resolution is configurable, please see [sampleSize Setting](#). You can either choose 16 bits which only have 13 bits resolution or 8 bit resolution which is Bluetooth Core Specifications Version 5.2 standard. It does not matter which resolution is chosen, each I/Q pair will always take 32 bits space in radio RAM.

The application layer passes in the antenna toggling table into RF Core, RF Core then does the antenna switching while collecting I/Q samples.

In the slave device, the RF Core ensures that the CTE is inserted at the end of the connection event packet without being distorted by the whitening filter.

In the passive and master devices, the RF Core analyzes the packet and starts capturing samples at the right time while synchronizing antenna switching. The samples are left in the RF Core RAM for analysis by the main MCU

## AoA Driver

For `rtls_passive`, an example that uses [Micro BLE Stack](#), the AoA driver is responsible for pin initialization, AoA enabling, data extraction and angle estimation.

AoA functionality in `rtls_master` is implemented as Bluetooth Core Specifications Version 5.2 compliant, therefore, the initialization, AoA enabling and data extraction are moved to host module. However, users can still use RTLS Control Module to setup the wanted parameters.



## Configurations Supported

- Sampling frequency supported: 1 Mhz (Bluetooth Core Specifications Version 5.2), 2 Mhz, 3 Mhz or 4 Mhz
- Sampling slot length supported: 1 us\* or 2 us (both are authorized by Bluetooth Core Specifications Version 5.2). *Note: Per BLE spec, the sampling slot and the switching slots must have the same duration.*
- Sample size supported: 8bit (Bluetooth Core Specifications Version 5.2) or 16bit

\* Sampling slot of 1us is supported by the AoA driver but some limitations exist when using the BOOSTXL-AOA hardware. The antennas of the BOOSTXL-AOA require 1.6 us to switch and settle (leaving 0.4 us in the sampling period to collect data). Other designs with faster RF switches can support 1us switching.

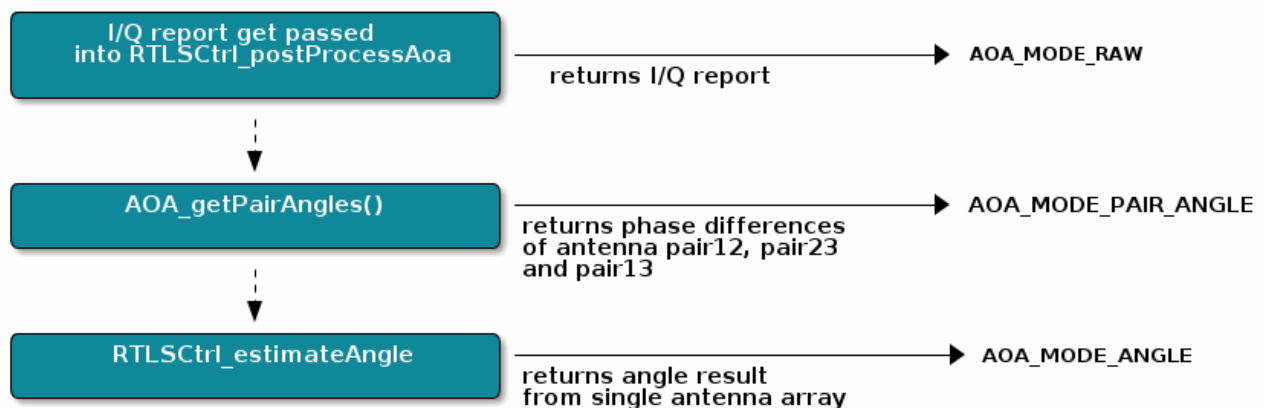
## Data Collection Flow

When RF Core detects AoA packets, it will start sampling the I/Q on the tone while toggling the antenna according to a user defined period.

For `rtls_passive`, IQ samples will be extracted after `RTLSPassive_monitorCompleteEvt()` has triggered the next sync event. During sync event processing `RTLSCtrl_postProcessAoa` will enable the radio RAM, and read out the IQ samples using `AOA_postProcess()`

For `rtls_master`, IQ samples will be sent to application from RTLService host module with event type `RTLSSRV_CONNECTION_CTE_IQ_REPORT_EVT`.

According to the AoA result mode set by python, different sets of functions will be called and will return the corresponding data set as illustrated below:



### ! Note

For `rtls_master`, when using the `AOA_MODE_RAW`, the BLE5-Stack will filter out the I/Q data from switching period if bit[0] in `sampleCtrl` is set to 0. To obtain the full set of I/Q data, you will need to set bit[0] in `sampleCtrl` to 1. Please see [sampleCtrl I/Q data Filter](#)

## Setting for the overview.

### Warning

For `rtls_master`, when `bit[0]` for `sampleCtrl= 1`, `AOA_MODE_PAIR_ANGLE` and `AOA_MODE_ANGLE` are not supported.

Result mode is set in the python as part of `RTLS_CMD_AOA_SET_PARAMS`.

Listing 118. `AoaSetParamsReq`

```
1 class AoaSetParamsReq(NpiRequest, SyncReq, FromAp):
2     command = Commands.RTLS_CMD_AOA_SET_PARAMS
3     struct = Struct(
4         "aoaRole" / Enum(Int8ul, AoaRole), # AOA_MASTER, AOA_SLAVE, AOA_PASSIVE
5         "aoaResultMode" / Enum(Int8ul, AoaResultMode), # AOA_MODE_ANGLE,
6         AOA_MODE_PAIR_ANGLES, AOA_MODE_RAW
7         "connHandle" / Int16ul,
8         "slotDurations" / Int8ul, # 1us/2us sampling slots
9         "sampleRate" / Int8ul, # 1Mhz (BT5.1 spec), 2Mhz, 3Mhz or 4Mhz - this enables
10        oversampling
11        "sampleSize" / Int8ul, # 8 bit sample (as defined by BT5.1 spec), 16 bit sample
12        (higher accuracy)
13        "sampleCtrl" / Int8ul, # sample control flags bit[0] = 0-default filtering,
14        bit[0]= 1-RAW_RF no filtering
15                                # bit 4,5 - 0x10 - ONLY_ANT_1, 0x20 - ONLY_ANT_2
16                                # 0x00 is not a valid option.
17        "samplingEnable" / Int8ul,
18        # 0 = mask CTE even if enabled, 1 = don't mask CTE, even if disabled (support
19        Unrequested CTE)
20        "numAnt" / Int8ul, # Number of antennas in antenna array
21        "antArray" / Int8ul[this.numAnt], # GPIO's of antennas
22    )
```

Table 22. `aoaResultMode` Setting

Value	Description
<code>AOA_MODE_ANGLE</code>	See before
<code>AOA_MODE_PAIR_ANGLES</code>	See before
<code>AOA_MODE_RAW</code>	See before

### Note

When using `AOA_MODE_ANGLE` the antenna pattern field must be 0,1,2 for when antenna array 1 is used or 3,4,5 when antenna array 2 is used.

### Warning

When using `AOA_MODE_RAW` mode, the sampling rate is fixed at 4MHz.

## Antenna Switching

Antennas toggle pattern is set in the python as part of `RTLS_CMD_AOA_SET_PARAMS`.

Listing 119. *AoaSetParamsReq*

```
1 class AoaSetParamsReq(NpiRequest, SyncReq, FromAp):
2     command = Commands.RTLS_CMD_AOA_SET_PARAMS
3     struct = Struct(
4         "aoaRole" / Enum(Int8ul, AoaRole), # AOA_MASTER, AOA_SLAVE, AOA_PASSIVE
5         "aoaResultMode" / Enum(Int8ul, AoaResultMode), # AOA_MODE_ANGLE,
6         AOA_MODE_PAIR_ANGLES, AOA_MODE_RAW
7         "connHandle" / Int16ul,
8         "slotDurations" / Int8ul, # 1us/2us sampling slots
9         "sampleRate" / Int8ul, # 1Mhz (BT5.1 spec), 2Mhz, 3Mhz or 4Mhz - this enables
10    oversampling
11         "sampleSize" / Int8ul, # 8 bit sample (as defined by BT5.1 spec), 16 bit sample
12    (higher accuracy)
13         "sampleCtrl" / Int8ul, # sample control flags bit[0] = 0-default filtering,
14    bit[0]= 1-RAW_RF no filtering
15                                     # bit 4,5 - 0x10 - ONLY_ANT_1, 0x20 - ONLY_ANT_2
16                                     # 0x00 is not a valid option.
17         "samplingEnable" / Int8ul,
18         # 0 = mask CTE even if enabled, 1 = don't mask CTE, even if disabled (support
19    Unrequested CTE)
20         "numAnt" / Int8ul, # Number of antennas in antenna array
21         "antArray" / Int8ul[this.numAnt], # GPIO's of antennas
22     )
```

Table 23. *Antenna Pattern*

Variable	Definition
numAnt	Number of IOs RF Core should toggle during I/Q sampling
antArray	The IOs should be toggled during I/Q sampling.

### ! Note

For `rtls_passive`, the `slotDuration`, `sampleRate`, `sampleSize`, `samplingEnable`, `numAnt` and `antArray` are hardcoded in the embedded software. The only configurable parameter is the `aoaResultMode`.

Following is an example provided in our `rtls_agent` toolbox.

Listing 120. *IO control pattern setup example*

```

1     if aoa:
2         if rtlUtil.is_aoa_supported(all_nodes):
3             aoa_params = {
4                 "aoa_run_mode": "AOA_MODE_ANGLE", ## AOA_MODE_ANGLE,
5                 AOA_MODE_PAIR_ANGLES, AOA_MODE_RAW
6                 "aoa_cc26x2": {
7                     "aoa_slot_durations": 1,
8                     "aoa_sample_rate": 1,
9                     "aoa_sample_size": 1,
10                    "aoa_sampling_control": int('0x10', 16),
11                    ## bit 0 - 0x00 - default filtering, 0x01 - RAW_RF no filtering,
12                    ## bit 4,5 - 0x10 - ONLY_ANT_1, 0x20 - ONLY_ANT_2
13                    "aoa_sampling_enable": 1,
14                    "aoa_pattern_len": 3,
15                    "aoa_ant_pattern": [0, 1, 2]
16                }
17            }

```

The example shows that 3 IOs( `aoa_pattern_len` ) should be toggled during the I/Q sampling and the order for toggling is set to 0, 1 and 2 which is mapped to the `antennaTbl[]` array element 0, 1 and 2 in `ble_user_config.c`. Bit[4:5] for `aoa_sampling_control` is set to 1 which corresponds to use only antenna array 1. For more information regarding antenna array selection, Please see [sampleCtrl Antenna Array Setting](#) for the overview.

The RF Core will then repeat the pattern until the end of the CTE. Afterwards, the IOs will be restored to their original state.

Even though RF Core takes care of the toggling, it does not initialize the pin state. The application should initialize the pin state by calling either `AOA_initAntArray` for `rtls_passive` or calling `RTLSSrv_initAntArray` for `rtls_master`. This is taken care when host sends `RTLS_CMD_AOA_SET_PARAMS`.

For users that want to use different IOs and patterns, please visit SimpleLink Academy -> RTLS Toolbox -> Angle of Arrival -> Task 3.

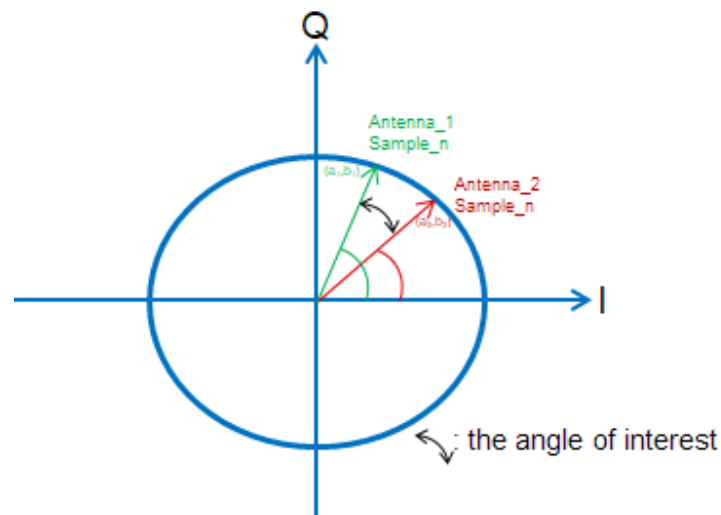
The RTLS API does not provide a direct way to determine which antenna has been used to sample a given piece of data. However, knowing the antenna switch sequence and the number of samples recorded per slot, it is possible to deduce the antenna used. Please see details in [Valid I/Q Samples For Angle Calculation](#).

## Convert I/Q Data to Angle Difference

When the radio frequency wave incident on to an antenna array (assuming there are only 2 antennas on the board, 4MHz sampling rate and 2us slot) and arrives at different antennas at different time, there will be a phase difference between the antennas. So we extract the phase difference between `ant1_sample[8 to 15]` and `ant2_sample[8 to 15]`. The switch among antennas will cause measurement error, therefore I/Q samples from 0 to 7 are discarded when calculating angles.

When using a custom HW with different antenna switch than what TI has on [BOOSTXL-AOA board](#), you might be able to use more I/Q samples if the antenna switch settling time is shorter than 1 us. The method for determining how many I/Q samples can be used is explained in [Valid I/Q Samples For Angle Calculation](#)

The I/Q data can be presented into a X-Y domain with real number I and imaginary number Q (90 degree difference). As mentioned before, for each period of 250 kHz signal, we sample 16 I and Q data. If there is no difference, that means that the I/Q data is the same, therefore the phase between ant\_1 sample1 will be the same to ant\_2 sample1.



$$\curvearrowright = \text{arcTan} [(a_1 + b_1j) (a_2 - b_2j)]$$

Here is the code putting I/Q data into 2 dimensions and then the I/Q pairs were passed into function `AOA_AngleComplexProductComp()` to get final angle.

*Listing 121. Get the phase differences.*

```

1 for (uint16_t r = 1; r < (numIqSamples -
2 (AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate))/(numAnt*sampleRate) : ++r) // Sample Slot

```

```

3  {
4  for (uint16_t i = 0; i < sampleRate; ++i) // Sample inside Sample Slot
5  {
6  // Loop through antenna pairs and calculate phase difference
7  for (uint8_t pair = 0; pair < numPairs; ++pair)
8  {
9  const AoA_AntennaPair *p = &gAoaReport.antConfig->pairs[pair];
10  uint8_t a = p->a; // First antenna in pair
11  uint8_t b = p->b; // Second antenna in pair
12
13  // Calculate the phase drift across one antenna repetition (X * complex conjugate
14  (Y))
15  int16_t Paa_rel =
16  AOA_AngleComplexProductComp(pI[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate +
17  r*numAnt*sampleRate + a*sampleRate + i],
18
19  pQ[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate + r*numAnt*sampleRate + a*sampleRate + i],
20
21  pI[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate + (r-1)*numAnt*sampleRate + a*sampleRate +
22  i],
23
24  pQ[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate + (r-1)*numAnt*sampleRate + a*sampleRate +
25  i]);
26
27  // Calculate phase difference between antenna a vs. antenna b
28  int16_t Pab_rel =
29  AOA_AngleComplexProductComp(pI[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate +
30  r*numAnt*sampleRate + a*sampleRate + i],
31
32  pQ[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate + r*numAnt*sampleRate + a*sampleRate + i],
33
34  pI[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate + r*numAnt*sampleRate + b*sampleRate + i],
35
36  pQ[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate + r*numAnt*sampleRate + b*sampleRate + i]);
37
38  // Add to averages
39  // v-- Correct for angle drift / ADC sampling frequency error
40  antenna_versus_avg[a][b] += Pab_rel + ((Paa_rel * abs(a-b)) / numAnt);
41  antenna_versus_cnt[a][b] ++;
42  }
43  }
44  }
45
46  int32_t AOA_AngleComplexProductComp(int32_t Xre, int32_t Xim, int32_t Yre, int32_t Yim)
47  {
48  int32_t Zre, Zim;
49  int16_t angle;
50
51  // X*conj(Y)
52  Zre = Xre*Yre + Xim*Yim;
53  Zim = Xim*Yre - Xre*Yim;
54
55  // Angle. The angle is returned in 256/2*pi format [-128,127] values
56  angle = AOA_iatan2sc((int32_t) Zim, (int32_t) Zre);
57
58  return (angle * angleconst);
59  }

```

Something to highlight is that in reality the 250 kHz might not be perfect (for example, could be 255kHz or 245kHz), therefore, there is slightly phase difference between ant\_1 sample\_n and ant\_1 sample\_(n + 16\*1). Therefore run time compensation is also applied:

Listing 122. Compensation method.

```

1 // Calculate the phase drift across one antenna repetition (X * complex conjugate (Y))
2 int16_t Paa_rel = AOA_AngleComplexProductComp(pI[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate +
3 r*numAnt*sampleRate + a*sampleRate + i],
4 pQ[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate +
5 r*numAnt*sampleRate + a*sampleRate + i],
pI[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate +
(r-1)*numAnt*sampleRate + a*sampleRate + i],
pQ[AOA_OFFSET_FIRST_VALID_SAMPLE*sampleRate +
(r-1)*numAnt*sampleRate + a*sampleRate + i]);

```

Because of the non-perfect 250 kHz tone, the phase difference is aggregated. Let's say that every period will have 45 degree of delay. Then when comparing ant\_1 sample\_n and ant\_1 sample\_(n+16\*1), the aggregated phase difference is 90 degree. But the real phase difference between every period is only 45 degree. Therefore the calculated phase difference must be divided by the number of antennas used, in this case 2.

Listing 123. Final phase difference

```

1 antenna_versus_avg[a][b] += Pab_rel + ((Paa_rel * abs(a-b)) / numAnt);

```

## Valid I/Q Samples For Angle Calculation

The BLE spec defines two terms "Switch slot" and "Sample slot". The length of the switch slot and sample slot must be equal, and only two values (1us or 2us) are allowed.

At the beginning of a CTE, the number of elements to discard depends on the length of the switch and sample slots and of the sampling frequency.

The following figure presents the way the CTE is sampled. The image is copied from the BLUETOOTH CORE SPECIFICATION Version 5.1 | Vol 6, Part B | 2.5.1.

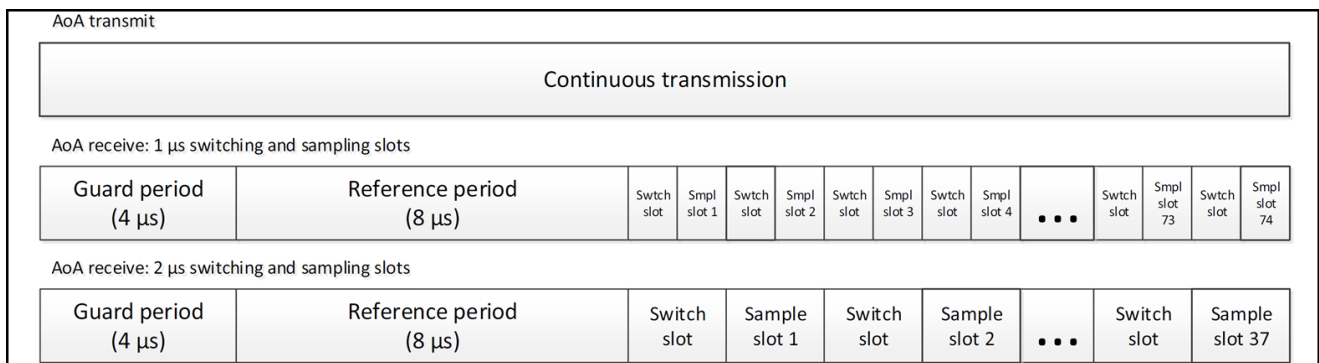


Figure 122. Constant tone sampling structure

- **When using 1us slots:**

- the guard period is 4us (no sampling is done during the guard period / no sample has to be discarded)
- the reference period is 8us. All the samples acquired during the reference period has to be discarded (e.g. if the sampling frequency is 4MHz, it means that 32 samples have to be discarded - 8 samples have to be discarded if sampling frequency is 1Mhz)
- the first switch slot (and all the others) lasts 1us. The data sampled during a switch slot has to be discarded (e.g. if the sampling frequency is 4MHz, it means that 4 samples have to be discarded during each switch slot - 1 sample per switch slot has to be discarded if sampling frequency is 1MHz).

As a result, when sampling frequency is 4MHz, the first sample to consider is the 37th. When sampling frequency is 1MHz, the first sample to consider is the 10th.

- **When using 2us slots:**

- the guard period is still 4us (no sampling is done during the guard period / no sample has to be discarded)
- the reference period is still 8us. All the samples acquired during the reference period have to be discarded (e.g. if the sampling frequency is 4MHz, it means that 32 samples have to be discarded - 8 samples have to be discarded if sampling frequency is 1Mhz)
- the first switch slot (and all the others) lasts 2us. The data sampled during a switch slot has to be discarded (e.g. if the sampling frequency is 4MHz, it means that 8 samples have to be discarded during each switch slot - 2 samples per switch slot have to be discarded if sampling frequency is 1MHz).
- **the sampling slots are divided in two**
  - one idle slot of 1us (which allows the antenna to settle). The data sampled during the idle slot has to be discarded (e.g. if the sampling frequency is 4MHz, it means that 4 samples have to be discarded during each idle slot - 1 sample per idle slot has to be discarded if sampling frequency is 1MHz).
  - one sample slot of 1us (e.g. if the sampling frequency is 4MHz, 4 samples are recorded per sample slot - 1 sample is recorded per sample slot if the sampling frequency is 1MHz)

As a result, when sampling frequency is 4MHz, the first sample to consider is the 45th. When sampling frequency is 1MHz, the first sample to consider is the 12th.

The following figure presents how each sample slot is divided. This image is copied from the BLUETOOTH CORE SPECIFICATION Version 5.1 | Vol 6, Part B | 2.5.4



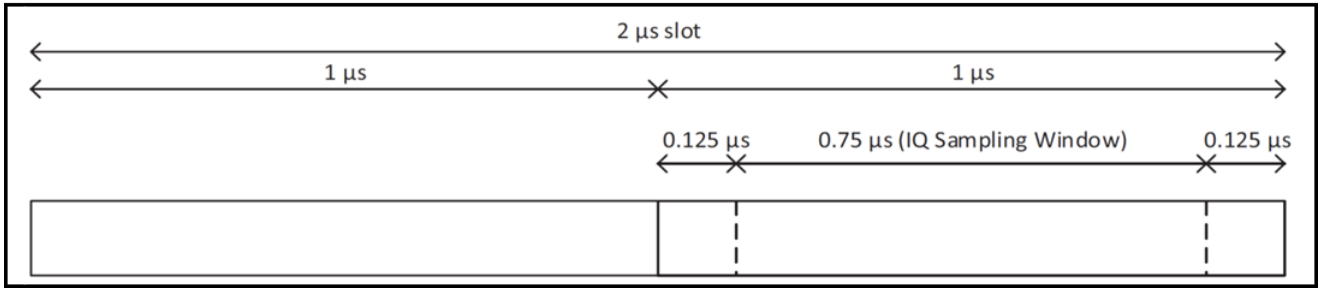
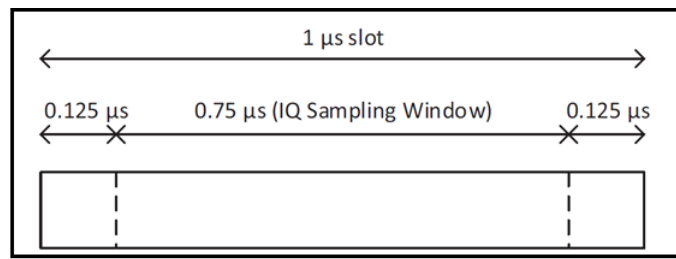


Figure 123. "Zoom" on a sampling slot

**Note**

It's important to note that 1us switch slot can only be used if the antennas are able to switch and settle in less than 1us. If the antennas are not able to switch and settle in less than 1us a part of the data sampled during the sampling slot won't be correct and will have to be discarded.

An other good way to determine what I/Q samples to use is to plot all the I/Q samples.

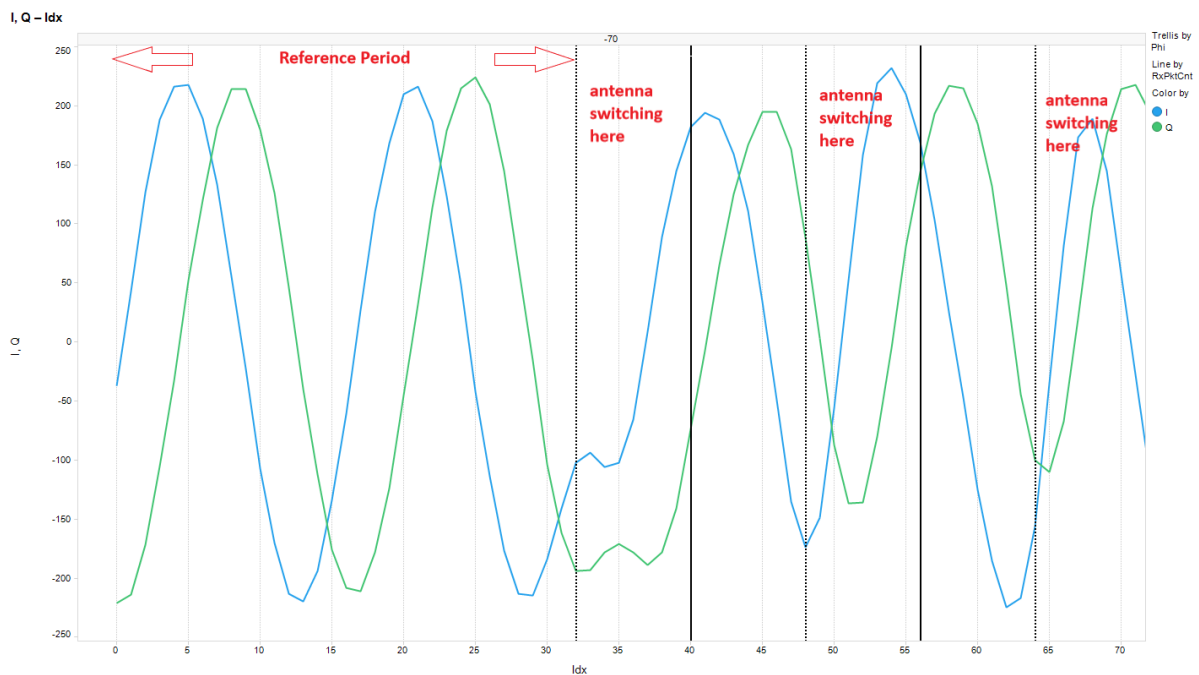
The picture below shows the I/Q samples which were collected using the `rtls_passive` example together with `rtls_master` and `rtls_slave` examples using 4MHz sampling rate and 2us slot.

Table 24. Axis description

Axis	Description
X top	Angle between a rtls_slave device and rtls_passive.
X bottom	Index number of I/Q data.
Y	I/Q values.

The first 32(index 0~31) samples are taken in reference period which there is no antenna switching. Therefore the I/Q plot looks like sinusoid wave.

After that at index 32, you can see when switching happened there comes discontinuity of I/Q samples.



### ! Note

It's easier to see the phase discontinuity when there is indeed phase difference. Therefore, before collecting I/Q data, make sure the angle between `rtls_passive` and `rtls_slave` is not 0 degree.

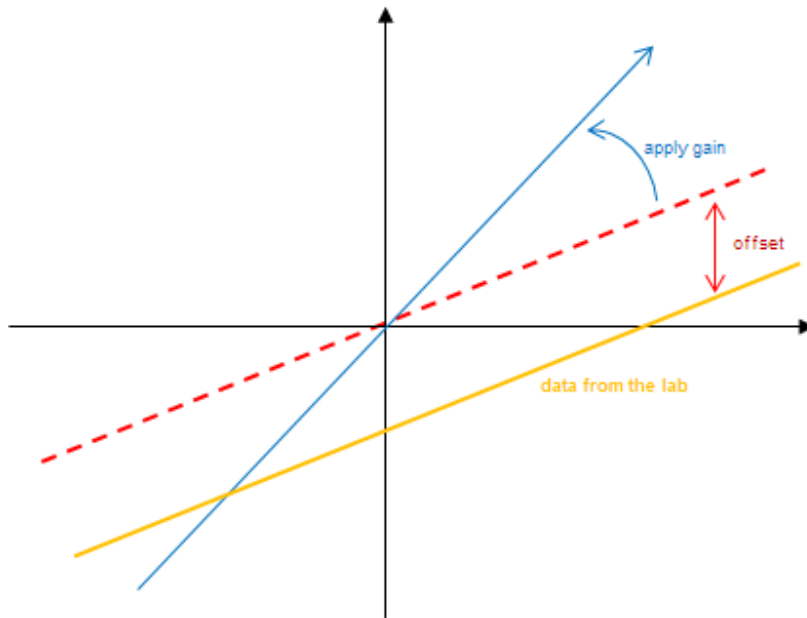
## Angle Compensation

Under `AoA_getPairAngles()`, the phase differences based on I and Q data were acquired. After that, angle compensation is added. Please see the code below. This is because angle estimation is affected by antenna pairs and frequency. The values `p->gain`, `p->offset`, `channelOffset_A1` and `channelOffset_A2` are based on lab measurements. Different antenna board design and frequency will give you different `p->gain`, `p->offset`, `channelOffset_A1` and `channelOffset_A2`.

The following code can be found in `AOA.c` `AoA_getPairAngles()`, this is antenna pairs compensation.

Listing 124. Antenna pair compensation

```
1 // Write back result for antenna pairs
2 for (int pair = 0; pair < numPairs; ++pair)
3 {
4     const AoA_AntennaPair *p = &gAoaReport.antConfig->pairs[pair];
5     gAoaReport.antResult->pairAngle[pair] = (int)((p->sign * antenna_versus_avg[p->a][p->b]
6 + p->offset) * p->gain);
7 }
```



As you can see from the image above, the offset is applied to make sure the data received at 0 degree will derive 0 degree after the calculation and then the slope is changed to make it fit better with all the rest of the angles.

The compensation values for antenna array 1 can be found in `ant_array1_config_boostxl_rev1v1.c` `AoA_AntennaPair pair_A1[]`

Listing 125. Values used for compensation

```
1  AoA_AntennaPair pair_A1[] =
2  {
3      // v12
4      .a = 0,           // First antenna in pair
5      .b = 1,           // Second antenna in pair
6      .sign = 1,        // Sign for the result
7      .offset = -5,    // Measurement offset compensation
8      .gain = 0.95,    // Measurement gain compensation
9  },
10 // v23
11 .a = 1,
12 .b = 2,
13 .sign = 1,
14 .offset = -20,
15 .gain = 0.9,
16 },
17 // v13
18 .a = 0,
19 .b = 2,
20 .sign = 1,
21 .offset = -20,
22 .gain = 0.50,
23 },
24 };
```

Followed by antenna pair compensation, we added frequency compensation. For antenna array 1, the values used for frequency compensation can be found in `ant_array1_config_boostxl_rev1v1.c` `int8_t channelOffset_A1[40]`.

Listing 126. Values used for compensation

```
1  int8_t channelOffset_A1[40] = {2, // Channel 0
2                                2, // Channel 1
3                                1, // Channel 2
4                                1, // Channel 3
5                                1, // Channel 4
6                                1, // Channel 5
7                                1, // Channel 6
8                                1, // Channel 7
9                                0, // Channel 8
10                               0, // Channel 9
11                               0, // Channel 10
12                               3, // Channel 11
13                               3, // Channel 12
14                               2, // Channel 13
15                               3, // Channel 14
16                               3, // Channel 15
17                               3, // Channel 16
18                               3, // Channel 17
19                               3, // Channel 17
20                               3, // Channel 18
21                               3, // Channel 20
22                               3, // Channel 21
23                               2, // Channel 22
24                               3, // Channel 23
25                               3, // Channel 24
26                               3, // Channel 25
27                               3, // Channel 26
28                               3, // Channel 27
29                               3, // Channel 28
30                               2, // Channel 29
31                               2, // Channel 30
32                               2, // Channel 31
33                               2, // Channel 32
34                               2, // Channel 33
35                               2, // Channel 34
36                               2, // Channel 35
37                               1, // Channel 36
38                               0, // Channel 37
39                               0, // Channel 38
40                               0, // Channel 39
41                               };
```

## AoA Functions Overview

Here is the list of the most important functions for AoA users.

### Python

1. `aoa_set_params` :

Listing 127. AoA Set Parameter

```

1  class AoaSetParamsReq(NpiRequest, SyncReq, FromAp):
2      command = Commands.RTLS_CMD_AOA_SET_PARAMS
3      struct = Struct(
4          "aoaRole" / Enum(Int8ul, AoaRole), # AOA_MASTER, AOA_SLAVE, AOA_PASSIVE
5          "aoaResultMode" / Enum(Int8ul, AoaResultMode), # AOA_MODE_ANGLE,
6          AOA_MODE_PAIR_ANGLES, AOA_MODE_RAW
7          "connHandle" / Int16ul,
8          "slotDurations" / Int8ul, # 1us/2us sampling slots
9          "sampleRate" / Int8ul, # 1Mhz (BT5.1 spec), 2Mhz, 3Mhz or 4Mhz - this enables
10         oversampling
11         "sampleSize" / Int8ul, # 8 bit sample (as defined by BT5.1 spec), 16 bit sample
12         (higher accuracy)
13         "sampleCtrl" / Int8ul, # sample control flags 0x0-default filtering, 0x1-RAW_RF
14         no filtering
15         "samplingEnable" / Int8ul,
            # 0 = mask CTE even if enabled, 1 = don't mask CTE, even if disabled (support
            Unrequested CTE)
            "numAnt" / Int8ul, # Number of antennas in antenna array
            "antArray" / Int8ul[this.numAnt], # GPIO's of antennas
        )

```

Table 25. slotDurations Setting

Value	Description
2	2us for antenna switching and 2us for I/Q sampling.
1	1us for antenna switching and 1us for I/Q sampling.

**Note**

For users that choose to have slotDuration = 1 needs to make sure that the RF switches on the custom board can settle down within 1 us.

Table 26. sampleRate Setting

Value	Description
4	Process packets with AoA present in the header and sample CTE at 4 MHz.
3	Process packets with AoA present in the header and sample CTE at 3 MHz.
2	Process packets with AoA present in the header and sample CTE at 2 MHz.
1	Process packets with AoA present in the header and sample CTE at 1 MHz.

Table 27. sampleSize Setting

Value	Description
-------	-------------

Value	Description
2	I/Q samples returned with 13 bits resolution.
1	I/Q samples returned with 8 bits resolution.

Table 28. sampleCtrl I/Q data Filter Setting

bit[0]	Description
1	BLE5-Stack returns whole I/Q data to the application.
0	BLE5-Stack filters out the I/Q data from switching period and returns the rest to appl

Table 29. sampleCtrl Antenna Array Setting

bit[4:5]	Description
0b01	Application use only antenna array 1.
0b10	Application use only antenna array 2.

### Note

rtls\_passive only supports the following configuration: `slotDuration` = 2, `sampleRate` = 4 and `sampleSize` = 2.

## RTLS\_Passive

The functions covered under this section are all in AOA.c.

1. `AOA_init`:

This function takes care of the IO initialization.

## AoA Applications Overview

### Note

The out of the box examples (`rtls_master` and `rtls_slave`) enable both connection and connectionless AoA. `rtls_passive` only enables connection AoA (connectionless AoA cannot be used on `rtls_passive`).

# Connection AoA

## Available tools for Connection AoA

Some Python scripts are provided in `<SDK>\tools\ble5stack\rtls_agent\examples`. The following Python scripts can be used with connection AoA:

- [Recommended to get started] `rtls_example_with_rtls_util.py`
- `rtls_aoa_iq_with_rtls_util_export_into_csv.py`
- `rtls_aoa_multi_conn_example.py`

The RTLS UI, provided in `<SDK>\tools\ble5stack\rtls_agent\rtls_ui` can be used with connection AoA.

All these tools require to attach to the computer one `rtls_master` and optionally one or several `rtls_passive` devices. The `rtls_slave` device(s) are not necessarily attached to the computer.

## Applications Overview

In the out of box software, the `rtls_master` and `rtls_slave` will form a BLE connection. After establishing the connection, `rtls_master` will send connection information through UART to PC and then the node manager will pass this piece of information to `rtls_passive` which can then track the connection.

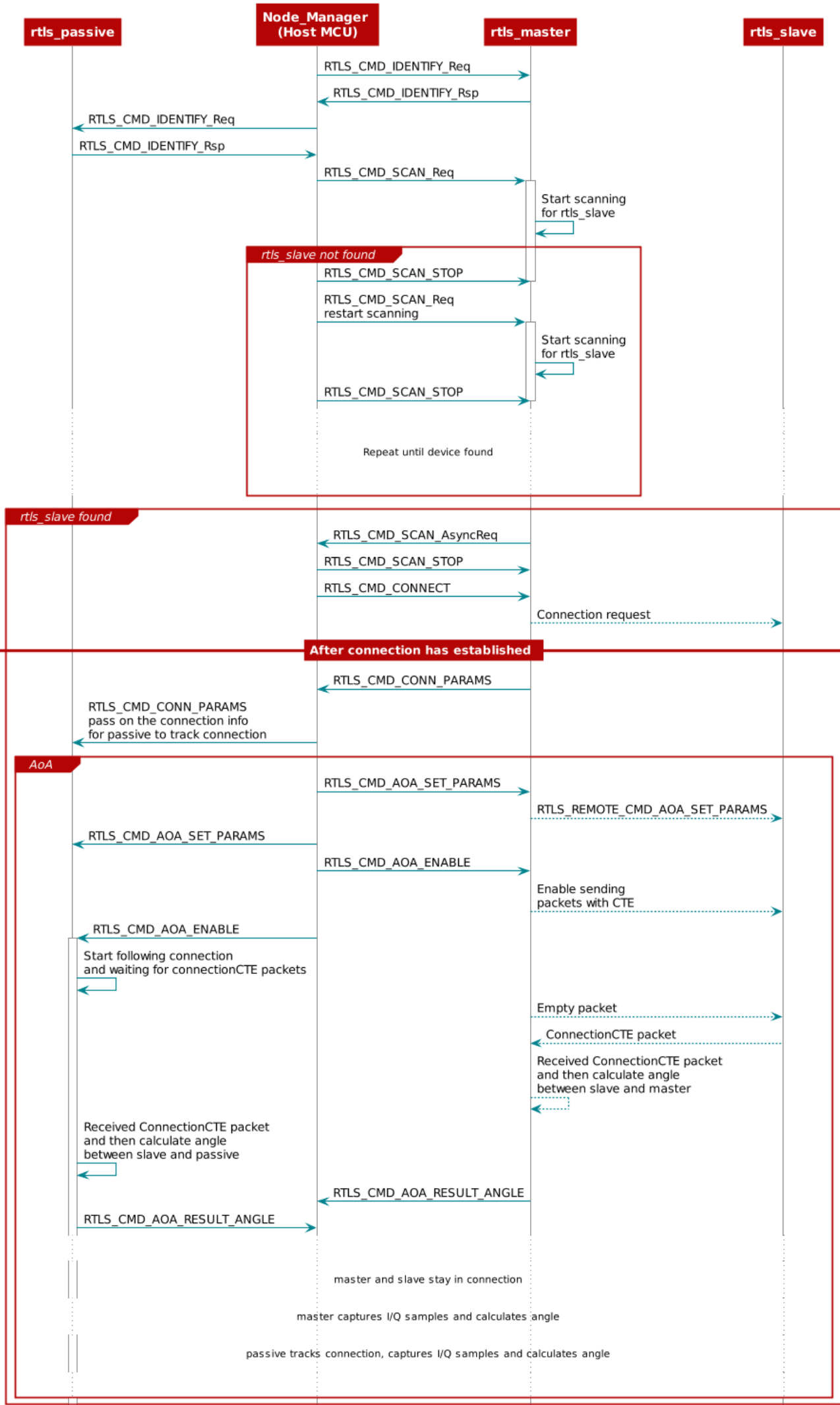
Next, the node manager sets up AoA parameters for `master` and `passive` and then `master` will send a packet over the air to slave to setup the CTETime.

After that, the `rtls_master` will send a start AoA request over the air to the `rtls_slave` and to `rtls_passive` over wire, then `rtls_slave` will append CTE at the end of every connection packet.

`rtls_passive` and `rtls_master` can then do I/Q sampling and calculate angles base on the ConnectionCTE packets.

The sequence diagram below illustrates the whole process of how out of box examples work.





## Connectionless AoA

In order to not flood the advertising channel, connectionless AoA is only allowed as Auxiliary packets (i.e. on secondary channels - see [Bluetooth Core Specifications Version 5.2 Vol 6, Part B, §2.3](#) for details on the PDUs allowed to be appended with a CTE). In our BLE5-Stack, we support connectionless AoA with periodic advertisement

packets. For more information regarding periodic advertisement, please see [Periodic Advertising](#).

### ! Note

In connectionless AoA there are no master/slave roles. There is the transmitter who sends the CTE over periodic advertisement packets and the receiver who synchronized with the advertiser and receives the CTE packets. The transmitter role is implemented by the device running the `rtls_slave` example. The receiver role is implemented by the device running the `rtls_master` example.

## Add / Remove Connectionless AoA support in `rtls_master` and `rtls_slave`

In the `rtls_master` project, support for reception and sampling of connectionless CTEs is enabled by defining the symbols `USE_PERIODIC_RTLS` and `USE_PERIODIC_SCAN`. These symbols are defined in the file `Tools\Defines\rtls_master_app.opt`.

In the `rtls_slave` project, support of sending of connectionless CTEs is enabled by defining the symbol `USE_PERIODIC_ADV`. This symbol is defined in the file `Tools\Defines\rtls_slave_app.opt`.

## Enable connectionless CTE transmission

The `rtls_slave` example project shows how to send periodic advertisements and append a CTE to them. First, the periodic advertisement has to be created and enabled (details are provided in [Periodic Advertising](#)). Once the periodic advertisement is enabled (i.e. the event `GAP_ADV_SET_PERIODIC_ADV_ENABLE_EVENT` is received with status `SUCCESS`), the CTE can be enabled using the function `RTLSSrv_SetCLCteTransmitParams`. Using this function, the length of each CTE and the number of CTEs to send per advertisement interval can be selected.

## Enable connectionless CTE reception

The `rtls_master` shows how to synchronize with a periodic advertisement and how to enable the reception of the CTE appended. The way to synchronize with a periodic advertisement is detailed in the GAP scanner section. After this, the function

`RTLSSrv_setCLCteSamplingEnableCmd` is used to start sampling the CTEs. The parameter `maxSampleCte` of this function determines how many CTEs the scanner can receive from a specific transmitter in each periodic advertisement interval.

### ! Note

The predefined symbol `MAX_NUM_CTE_BUFS` sets the maximum value that can be defined for `maxSampleCte`. By default, `MAX_NUM_CTE_BUFS` is set to 1 and is defined in the file `Tools\Defines\rtls_slave_app.opt`. The amount of memory allocated to connectionless CTE sampling is given by `MAX_NUM_CTE_BUFS * 2500` bytes.

## Available tools for Connectionless AoA

The Python script `rtls_connectionless_aoa_example_with_rtls_util.py` provided in `<SDK>\tools\ble5stack\rtls_agent\examples` can be used with connectionless AoA.

The RTLS UI, provided in `<SDK>\tools\ble5stack\rtls_agent\rtls_ui` can be used with connectionless AoA.

All these tools require attaching to the computer one `rtls_master`. The `rtls_slave` device(s) is (are) not required to be attached to the computer.

## Applications Overview

In the out of box software, the `rtls_slave` sends both legacy and periodic advertisements. Legacy advertisements are used by the `rtls_master` in the case of connection AoA to establish a connection (this point is not discussed in this section). In the case of connectionless AoA, only the periodic advertisements are used. These periodic advertisements are appended with a CTE.

The `rtls_master` device scans to find advertisements. The `rtls_master` forwards the advertisements found to the host MCU. Both legacy and periodic advertisements are reported back to the host MCU. The host MCU selects the type of advertisements to consider, hence the AoA type to perform.

In the case of connectionless AoA, the host MCU selects the periodic advertisement the `rtls_master` has to synchronize with.

The `rtls_master` synchronizes with the periodic advertisements sent by the `rtls_slave`. The `rtls_master` can then do I/Q sampling and calculate angles based on the CTE appended to the periodic advertisements.

As a reminder, the `rtls_passive` is not involved for connectionless AoA.

The sequence diagram below illustrates the whole process of how out of box examples work.

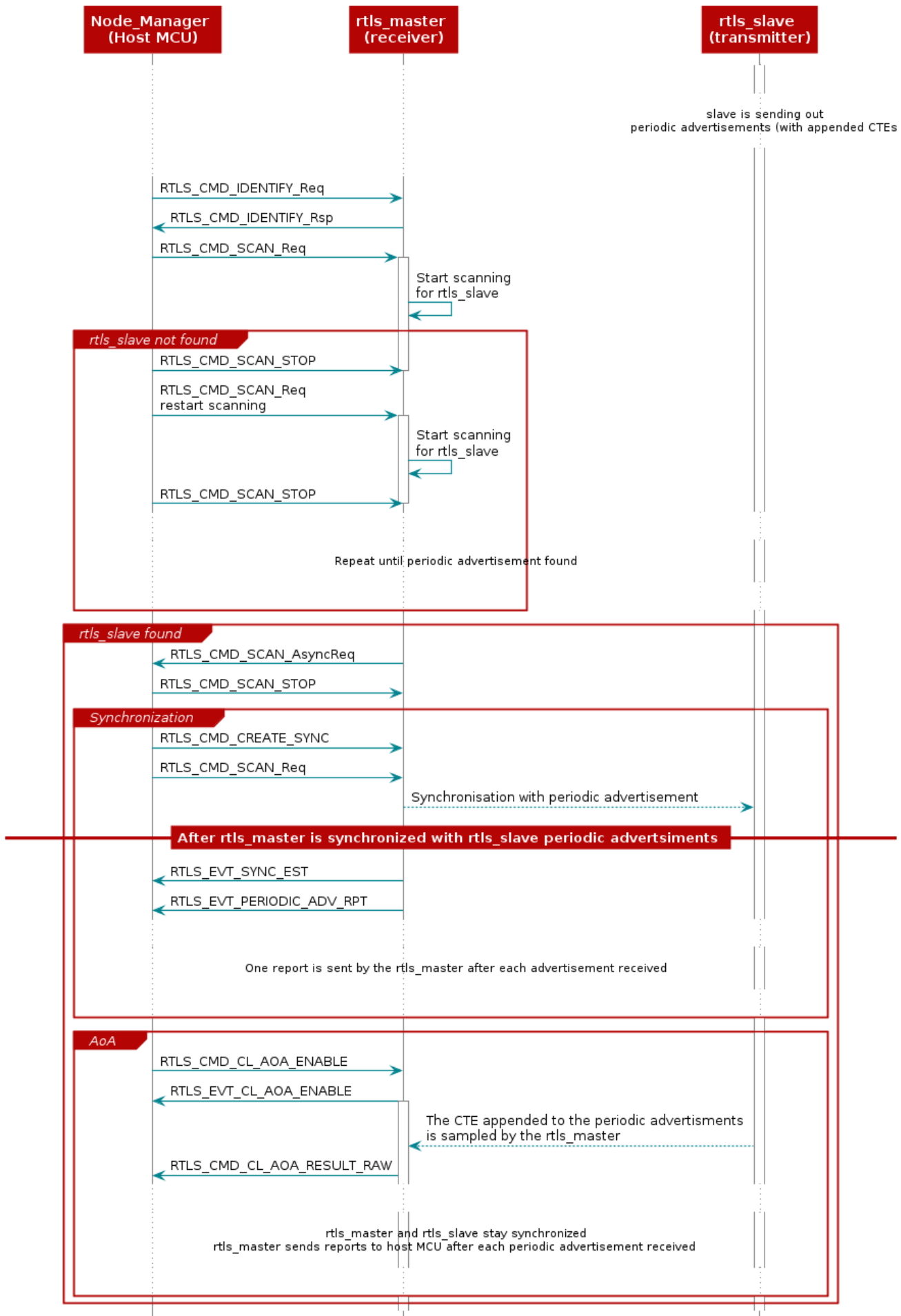


Figure 125. Setting up RTLS Connectionless AoA and enable AoA

