ECEN 5823-001
Internet of Things Embedded Firmware

Lecture #12

04 October 2018

Be Boulder.

University of Colorado Boulder

Agenda

- Class Announcements
- Bluetooth Low Energy / Smart
- Firmware best practices

Class Announcements

- Quiz #6 is due at 11:59 on Sunday, October 6th, 2018
- Homework #4: BLE Server + MITM + LCD due on Sunday, October 7th, at 11:59pm
- Homework #5: Client flowchart due on Wednesday, October 10th, at 11:59pm
- Homework #6: Client-Server due on Wednesday, October 17th, at 11:59pm

BLE: Central (Connecting to Devices)

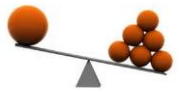
- An example of a compromise would be a connection interval of 15mS to 30mS and a slave latency of 150mS
  - Allows both rapid collection of data about the peripheral using up to >60Hz connection intervals and a possible slave idle frequency of >6Hz
- The slave might request different parameters from those that an application on the central device has chosen after connection has been made
  - For example:
    - The peripheral may request a connection interval of 150mS and a slave latency of 750mS to minimize energy use
    - But, the application on the central device might require the data or state from the peripheral every 50mS, so the central sets up connection interval of 50mS and slave latency set to 0
    - The application will get the data it requires at the appropriate data rate, but the peripheral battery life will be negatively impacted

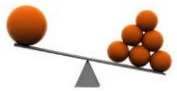
BLE: Central (What does this device do?)

- After connecting to the peripheral, the central device will need to discover what the device does using the following four procedures:
  - Primary Services Discovery
  - Relationship Discovery
  - Characteristic Discovery
  - And, Descriptor Discovery
- The first process is the Primary Services Discovery
  - These are the services that describe what the device does
  - For example:
    - If the device has a battery, the primary services would expose the Battery Service
    - If the device has a temperature sensor, the primary services would expose the Temperature Service
    - If the device had a temperature sensor within the battery, this secondary service of the battery would not be exposed through Primary Services Discovery

BLE: Central (What does this device do?)

- Next, for each Primary Service that the central device knows could include another service (Relationship Discovery)
  - This is where the secondary service of a temperature sensor in a battery would be discovered
- Does the set of services that a device have determine the set of profiles that the peripheral device supports?
  - The set of services that a device does not necessarily determine the set of profiles that the peripheral device supports.
  - Due to the complex algorithm to match the profile to the services provided, it is the more resource rich device, the central, that matches the profile to the peripheral
  - The benefit of this approach is that future client profile roles that use a set of services on a peripheral do not need to be designed into the peripheral when manufactured





BLE: Central (What does this device do?)

- After the primary and secondary services are discovered, the central devices enter the Characteristic and Descriptor phases
  - There is no revision numbers in BLE
  - Therefore, the only way to know if a given optional feature exists is to check for the exposure of a given characteristic that is linked to the optional feature

BLE: Central (Interacting with Services)

- Once the central device has completed its connection with the peripheral and discovered its services, it can begin to read and write characteristics and descriptors
- The protocol used to perform the reads and writes is the Attribute Protocol
  - The protocol has no state when connected or between one connection and the next
  - Any state that is maintain would be maintained in the “application layers of the BLE stack”
- Read Characteristics:
  - The most basic of all services is simply exposing a set of readable characteristics
  - For example, the temperature value of a temperature sensor

BLE: Central (Interacting with Services)

- Writeable Characteristics:
  - The next level of complexity is a service that has characteristics that is both writeable and readable
  - For example, the Link Loss Service is a writeable characteristic. An Alert Level can be written by the client to configure the behavior when the link between two devices is lost
    - If the client writes “No Alert” into the characteristic, then when the devices disconnect, the server will do nothing
    - If the client writes “Mild Alert” into the characteristic, then when the devices disconnect, the server will use a mild alert to notify the user
    - If the client writes “High Alert” into the characteristic, then when the devices disconnect, the server will alert the user all possible means that the server is configured

BLE: Central (Interacting with Services)

- Control Points:
  - A control point is a type of service that the client can write to, but it has no “state”
  - The control point has no “state” due to the service uses the written value immediately, and the server does not have any need to store that value after it has been consumed.
  - For example:
    - A client may want a peripheral to sent out an alert immediately
    - The client could set the characteristic in the Link Loss Service to “Mild” or “High Alert” and disconnect which would then trigger the server to perform the alert
    - But, the client may not want to disconnect
    - Instead, the client could write to the Alert Level characteristic in the Immediate Alert Service
    - Immediate Alert Service is only writeable and causes an alert immediately
    - With the characteristic consumed immediately, the alert sounded, there is no reason to save state

BLE: Central (Interacting with Services)

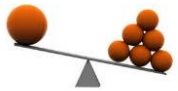
- State Machines:
  - The state machine exposes writeable control points and readable characteristics
  - These expose the state of the state machine
  - The difference between a state machine and a control point is that state is remembered in a state machine
  - Example: A state machine for synchronization
    - It would have two states
      - The machine is doing nothing (Idle)
      - And, the machine is searching for an accurate time (Searching)
    - To control the state machine, control points are made available such as
      - Start Synchronization
      - Cancel Synchronization

BLE: Central (Interacting with Services)

- State Machines:
  - Example: A state machine for synchronization (continued)
    - The state machine should be well defined
      - If in Idle and the control point Start Synchronization, go to Search
      - If in Search and the control point Cancel Synchronization, go to Idle
      - If in Idle and the control point Cancel Synchronization, stay in Idle (Do Not go into Error)
      - If in Search and control point Start Synchronization, stay in Search (Do Not go into Error)

BLE: Central (Notifications and Indications)

- Services expose state. Some change infrequently or random
- Polling these services would consume energy on the resource starved peripheral by transmitting state information that is not changing
  - Example: The battery state of a peripheral may change only once a day, but polling it every 15 minutes would increase the drain on the battery
    - If the battery is polled only once a day, the client may think the battery is full even though the battery may be out of charge



- A better process is to set up a notification
  - The client can configure the desired service's characteristics to send a notification as required
  - Most notifications are defined by the service, but some can be configured further by the client writing to additional characteristic descriptors
  - The notification will be sent during the next connection event between the client and peripheral
  - Not acknowledgement from the master is required

BLE: Central (Notifications and Indications)

- Indications are similar to notifications, but a confirmation message from the client is sent back to the server that it has received the data and that the application has received the data

BLE: Attributes background

- Definition per Google: a piece of information that determines the properties of a field or tag in a database or a string of characters in a display
- In Bluetooth Smart, a basic concept is that some devices have data and others want to use it
- The device that has the data is the server, and the device that wants to use it is the client

BLE: Attributes background

- Data is a value that represents something such as fact or measurement
  - Examples include the temperature of a room as measured by a thermometer or read by a heating system. Multiple devices can “know” data
- State is a value that represents the status or condition of a device such as what it is doing or how it is operating
  - State is only know by one device, so only one device can hold state information
- For our Bluetooth Smart discussions, “state” will refer to information (data) that resides on the server and that “data” refers to information as it is in transit from the server to the client and held on the client
- The data on the client is not authoritative because the server’s state could have changed since the client last received it

BLE: Attributes background

- Bluetooth Smart uses three different kinds of state:
  - External state is measured by an external sensor and that every time that it is read, it may change such as a temperature sensor

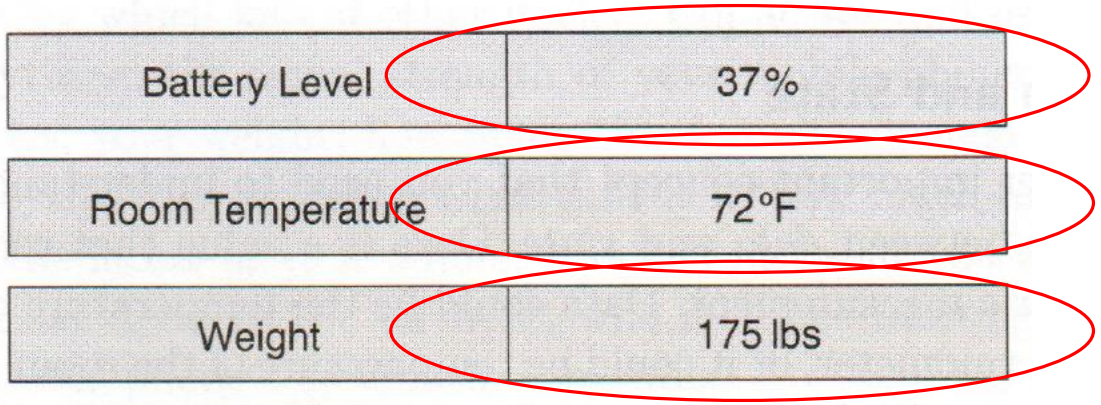


Figure 10–3 Physical measurements

BLE: Attributes background

- Bluetooth Smart uses three different kinds of state:
  - Internal state corresponds to the state of a state machine that represents the internal state of the device

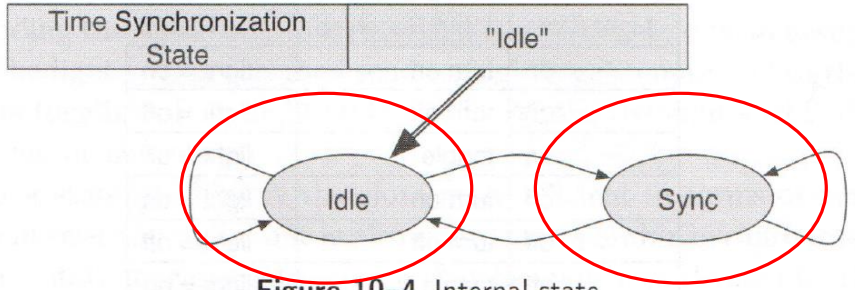


Figure 10–4 Internal state

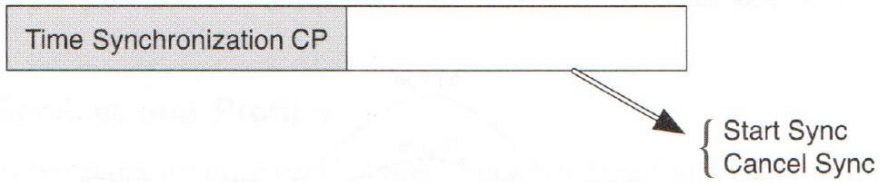


Figure 10–5 Abstract state

BLE: Attributes background

- Bluetooth Smart uses three different kinds of state:
  - Abstract state is state information that is only relevant at a momentary point in time and it does not represent external or internal state of the device

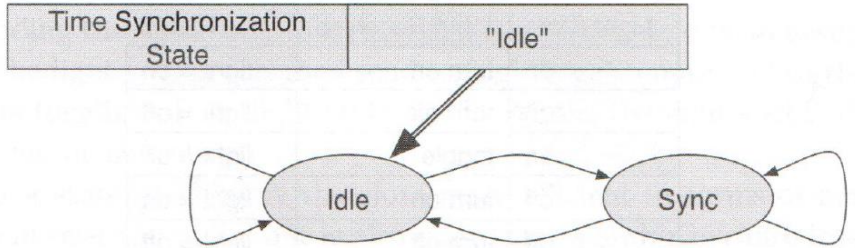


Figure 10–4 Internal state

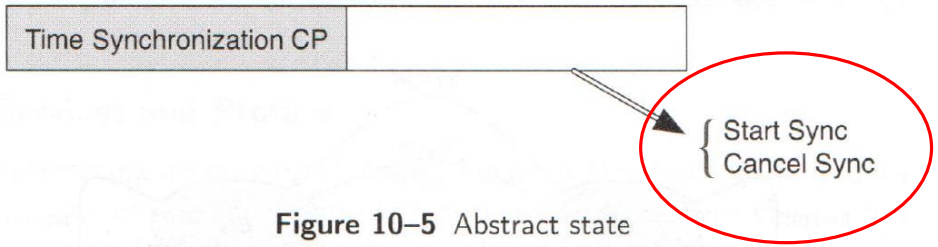


Figure 10–5 Abstract state

Examples of an abstract state are control points. They are commands that tell a device to perform an action, but in itself it has not state.

BLE: Attributes background

- A server's behavior is defined in a service specification
- The server specification defines the state that is exposed by the server using an attribute database
- Attributes on a service may be readable returning historical or current data while some attributes may be writeable to send commands (control points) to the server
- The profile specifications define how to use one or more services to enable a given use case

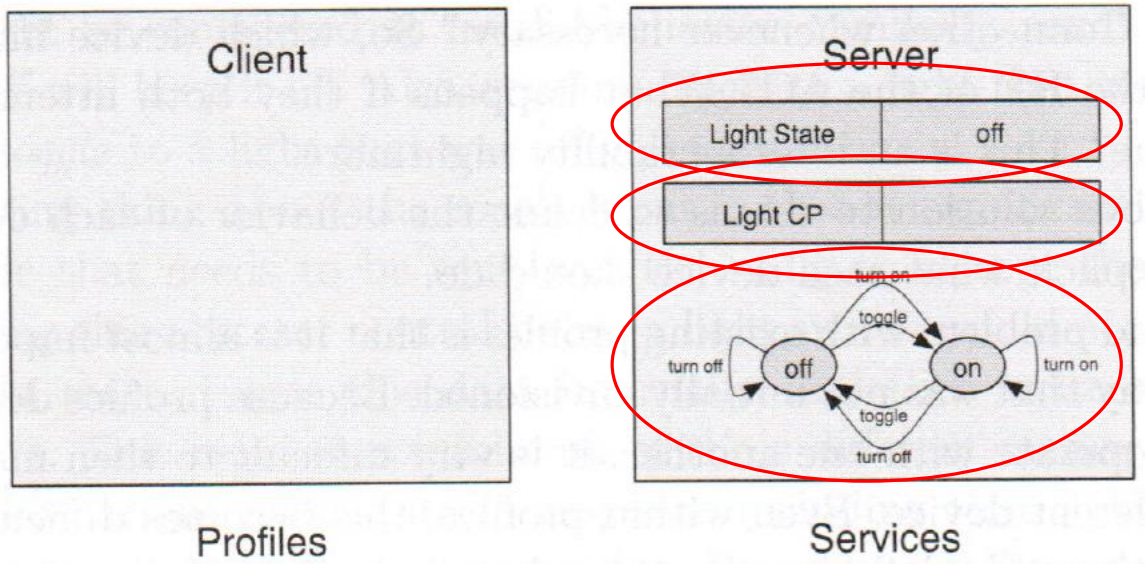


Figure 10–8 The profile/service architecture

BLE: Attributes background

- The profile specifications define how to use one or more services to enable a given use case
- As an example, how to configure the attributes exposed for the service in an attribute database on the server to ask the server to do something that the client needs it to do
- Defining specifically the server's services and independent of the client enables the service to be independently tested and independent of a client
- Any client that is given access, a connection, to the service, can use the service

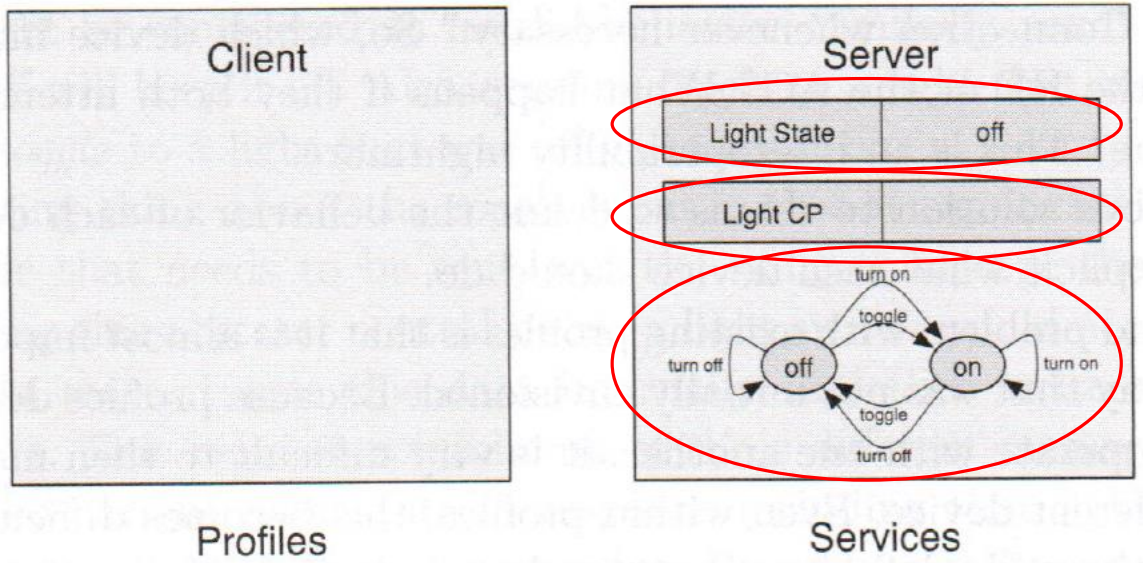


Figure 10–8 The profile/service architecture

BLE: Attributes (background)

- A client's behavior is defined in a profile specification
- As an example, a light switch would implement the light profile that knows how to control the light state through the server attributes and control points
- The light switch, the client, knows the behavior of the light service because the server specification, Light Service, to be the same for every instance
- The client profiles are essentially a set of rules for discovering, connecting, configuring and using a service

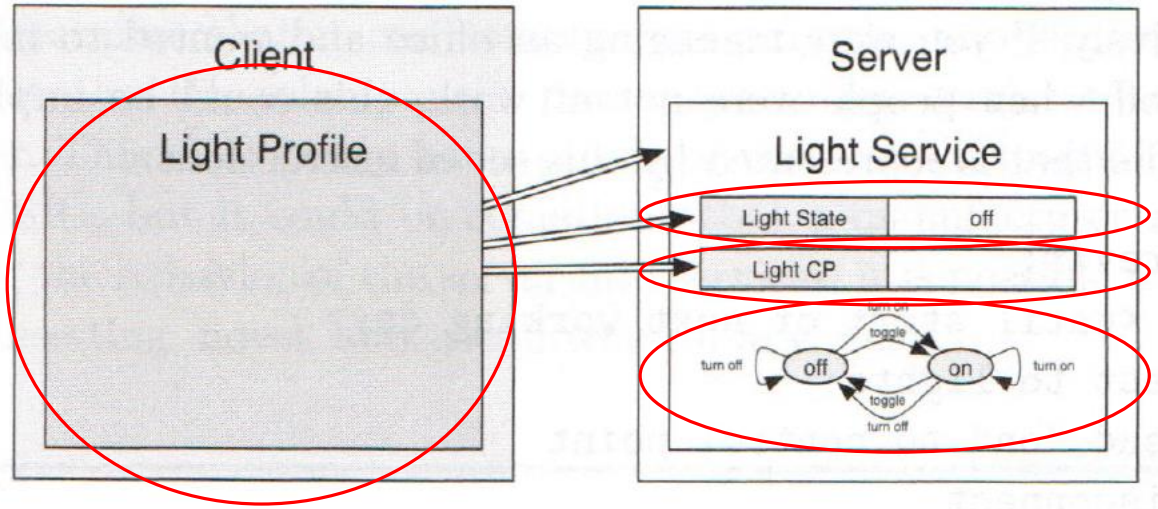


Figure 10–9 An example of a light profile and service

BLE: Attributes background

- Example 1: A home security system that knows that if the house is unoccupied and the homeowners would like the house look like it was occupied, a simple client profile could be written to the light service

  Loop forever:

  Wait <random period from 10 seconds to 3 hours>

  Connect to a light:

  Send “toggle” to control point

  disconnect

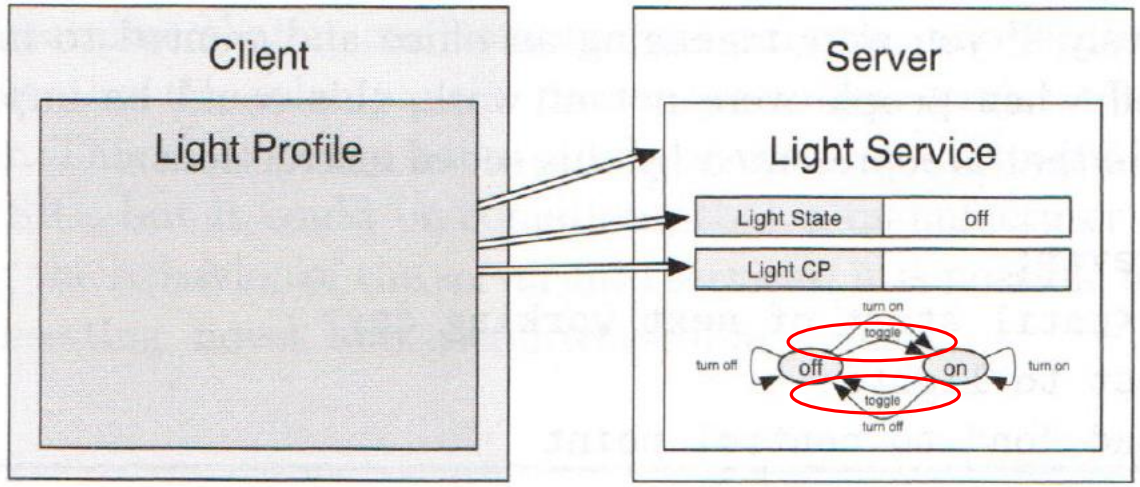


Figure 10–9 An example of a light profile and service

BLE: Attributes background

- Example 2: Take the same light service Server and install it in an office. In this example, the office managers would like the lights to be on while the staff is in the office

Loop forever:

Wait <until start of next working day>

Connect to lights:

Send “on” to control point

Disconnect

Wait <until end of work day>

Connect to lights:

Send “off” to control Point

Disconnect

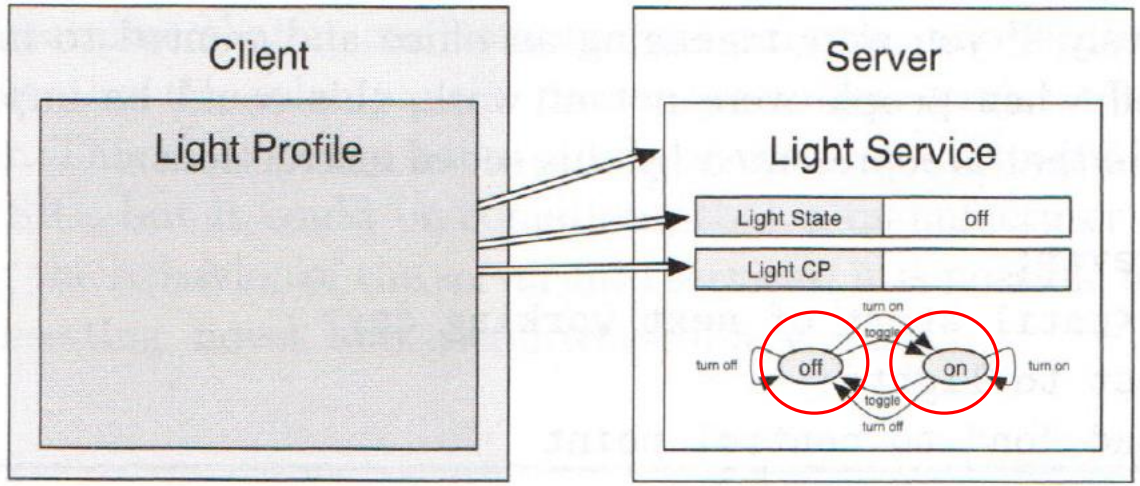


Figure 10–9 An example of a light profile and service

BLE: Attributes background



- The powerful combinations of the different services within the client profile make Bluetooth Low Energy. In the previous examples the client profile could be a mixture of services of occupancy sensors, time of day, and light services.
- Each individual service can be kept very simple. For this model to work, the services must be atomic
- Atomic in this context means that services perform only one set of actions. Make the services atomic make the services available to different clients as well as different client profiles.
  - If services were not atomic, the client profile previously may not have had access to the occupancy or timer services available in a device
- Another key concept is that the services are not dependent on each other which means any combination of services is possible

Review of BLE states

- What are example of internal states?
- What are example of external states?
- What are examples of abstract states?

BLE: Attributes

- An Attribute is a piece of labeled, addressable data
- The Bluetooth Smart attribute is composed of three values:
  - Attribute handle:
  - Attribute type:
  - Attribute value:

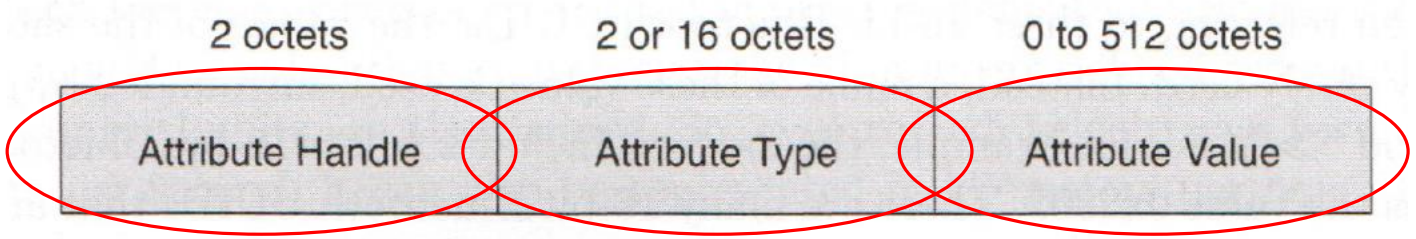


Figure 10–10 The structure of an attribute

BLE: Attributes

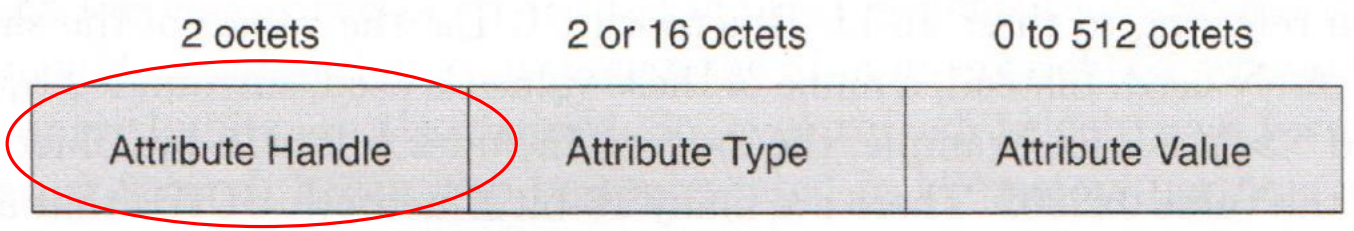


Figure 10–10 The structure of an attribute

- The attribute handle can be considered the memory address of the attribute
- For example, a device could have two temperature sensors which would have the same attribute type
  - To get the attribute value for the first temperature sensor you would need to access it through its attribute handle.
  - Similarly, to read the second temperature sensor you would need to read it through its own attribute handle.

BLE: Attributes

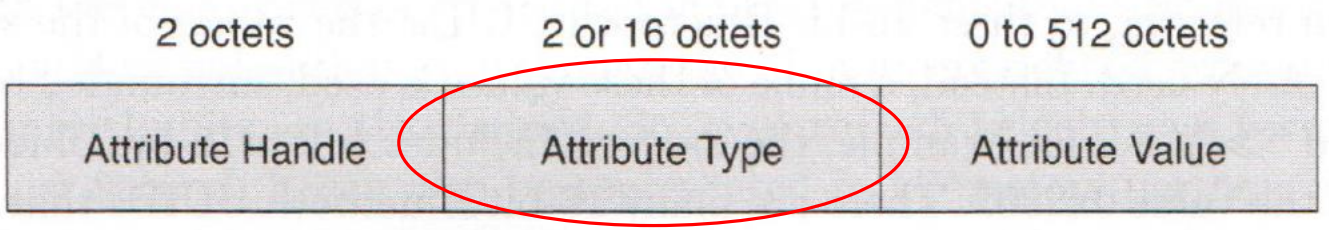
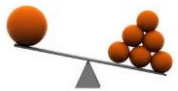


Figure 10–10 The structure of an attribute

- The attribute type is the type of data that the attribute value exposes such as temperature, pressure, power, time, etc.
- To address all the possible attribute types, a 128-bits is used for the attribute type which is a called a Universally Unique Identifier (UUID)
- 128-bits, 16-bytes, to address every attribute type would be very energy expensive to transmit on the BLE radio. For the most common attributes, there is an abbreviated 16-bit, 2 byte, UUID which is combined with the Bluetooth Base UUID to provide the complete 128-bit UUID



$$00002A01 - 0000 - 1000 - 8000 - 00805F9B34FB$$

BLE: Attributes

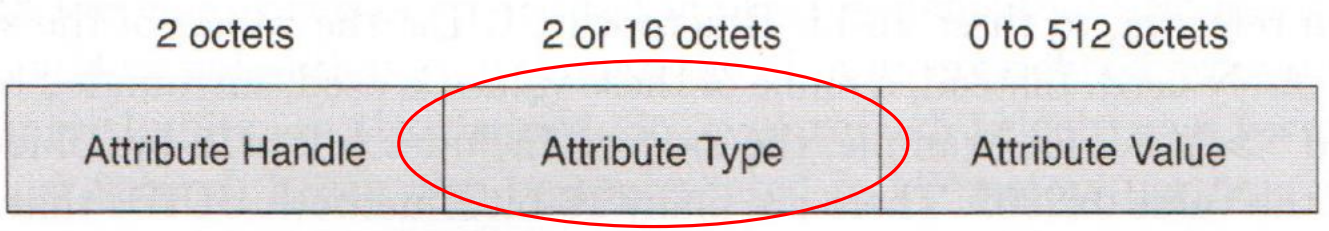


Figure 10–10 The structure of an attribute

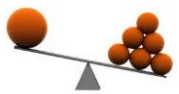
- When referring to the 16-bit UUIDs, instead of the actual 16-bits, for readability, they are named inside these brackets << >>
  - Examples: <<Include>> refers to the BLE 16-bit UUID 0x2802
- The 16-bit UUIDs are arranged in the following groups for readability:

0x1800 through 0x26FF are for Service UUIDs
0x2700 through 0x27FF are for Units
0x2800 through 0x28FF are for Attribute Types
0x2900 through 0x29FF are for Characteristic Descriptors
0x2A00 through 0x7FFF are for Characteristic Types

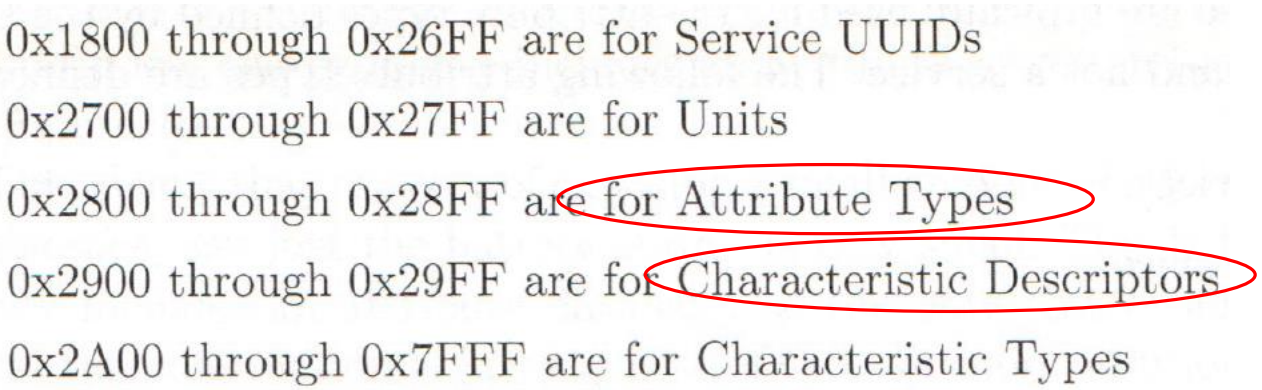
BLE: Attributes

0x1800 through 0x26FF are for Service UUIDs
0x2700 through 0x27FF are for Units
0x2800 through 0x28FF are for Attribute Types
0x2900 through 0x29FF are for Characteristic Descriptors
0x2A00 through 0x7FFF are for Characteristic Types

- Each BLE service can be identified by a UUID. The 16-bit shorten UUID allows 3,840 unique services to be addressed
- Many of the attribute values can be represented in different units such degree Celsius or Fahrenheit.
  - The Units 16-bit UUID are derived from the Bureau International des Poids et Mesures or otherwise known as the International System of Units
  - The Unit field enables the client to take the attribute value and appropriately apply it to the profile. For instance, if the attribute was the velocity of a car, the client profile may require the value in MPH or KPH. Knowing what the attribute value is, the client can convert it to the profile's input requirements if required

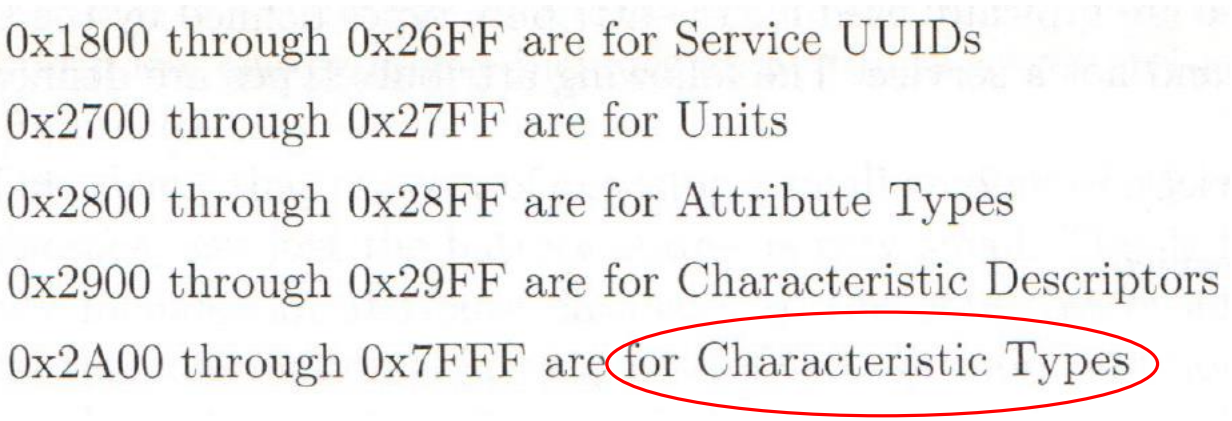


BLE: Attributes



- The Attribute Type 16-bit UUIDs are the most fundamental attribute types.
  - These are typically used for the attributes types defined by the Generic Profile, and not a service.
    - Primary Service
    - Secondary Service
    - Include
    - Characteristic
- Some data exposed by a service may require additional data. The additional data is described by using Characteristic Descriptors

BLE: Attributes



- Each unique type of value that is exposed to a service is allocated a characteristic type
  - The characteristic types 16-bit UUIDs enables 22,015 different characteristic types to be defined
  - This enables a client to discover all the different types of data that a server has
  - Each characteristic type has a defined format and representation

BLE: Attributes

- A collection of attributes is called a database
  - A database can be very small, a minimum of 6 attributes
  - Or very large and complex
    - The complexity is not at the attribute layer, but how those attributes are used in services and profile
  - The attribute database is always contained on the attribute server
    - An attribute client uses the Attribute Protocol to communicate with the attribute server
  - With only one attribute server on a device, there is only one attribute database
  - For a BLE device, the attribute database includes a Generic Access Profile service that is mandatory to support
  - Since both the client and server require the Generic Access Profile, every BLE device includes an attribute server and database

BLE: Attributes

- Attribute database example
  - GAP Service Attribute
  - TX Power Service
  - Battery Service

Attribute Handle	Attribute Type	Attribute Value
0x0001	Primary Service	GAP Service
0x0002	Characteristic	Device Name
0x0003	Device Name	"Proximity Tag"
0x0004	Characteristic	Appearance
0x0005	Appearance	Tag
0x0006	Primary Service	GATT Service
0x0007	Primary Service	Tx Power Service
0x0008	Characteristic	Tx Power
0x0009	Tx Power	–4dBm
0x000A	Primary Service	Immediate Alert Service
0x000B	Characteristic	Alert Level
0x000C	Alert Level	
0x000D	Primary Service	Link Loss Service
0x000E	Characteristic	Alert Level
0x000F	Alert Level	"high"
0x0010	Primary Service	Battery Service
0x0011	Characteristic	Battery Level
0x0012	Battery Level	75%
0x0013	Characteristic Presentation Format	uint8, 0, percent
0x0014	Characteristic	Battery Level State
0x0015	Battery Level State	75%, discharging
0x0016	Client Characteristic Configuration	0x0001

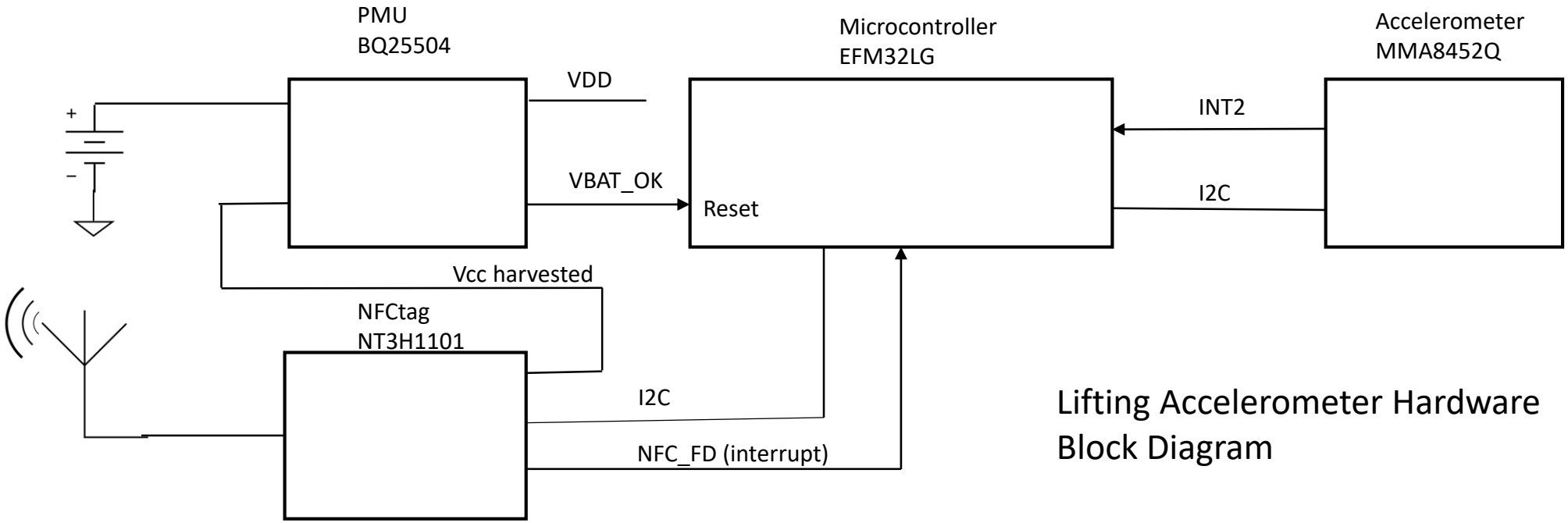
Figure 10–11 An example of an attribute database

Firmware best practices

- Planning up front will
  - Create the firmware scope of the project
  - Identify potential risks up front
  - Develop more “solid” code
  - Aid in the eventual debug of the hardware and firmware
- Like hardware, planning starts with developing a software “block” diagram
  - Software architecture block diagram
  - Organizational diagram of the software architecture
  - And, layered software architecture diagram

Firmware best practices

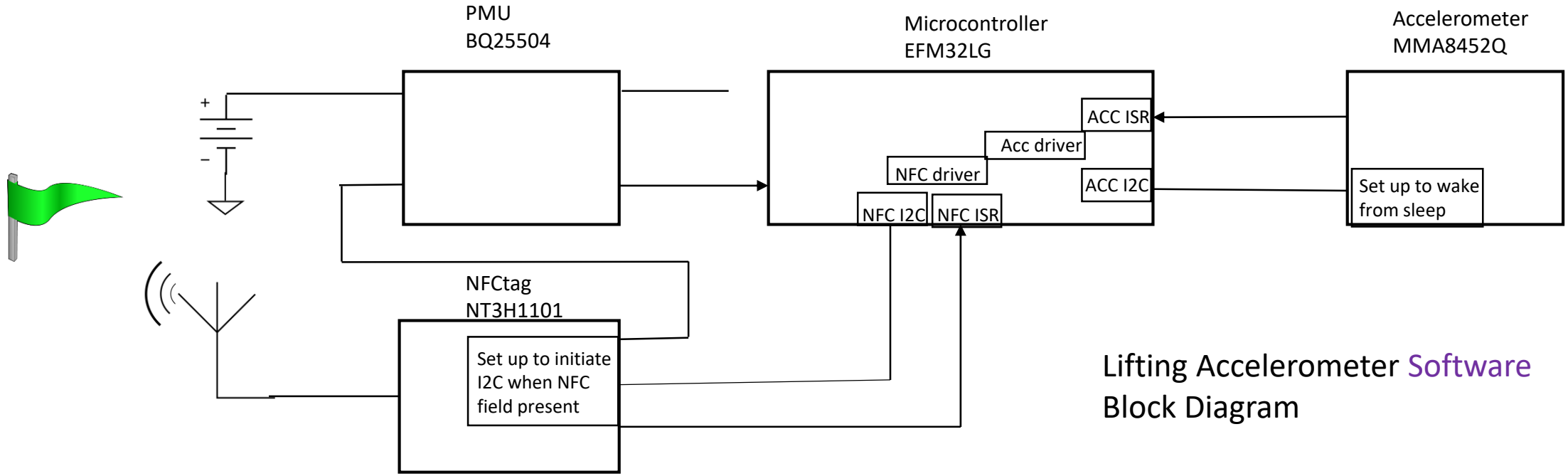
- The software block diagram starts from the hardware block diagram



Lifting Accelerometer Hardware Block Diagram

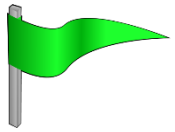
Firmware best practices

- The software block diagram starts from the hardware block diagram

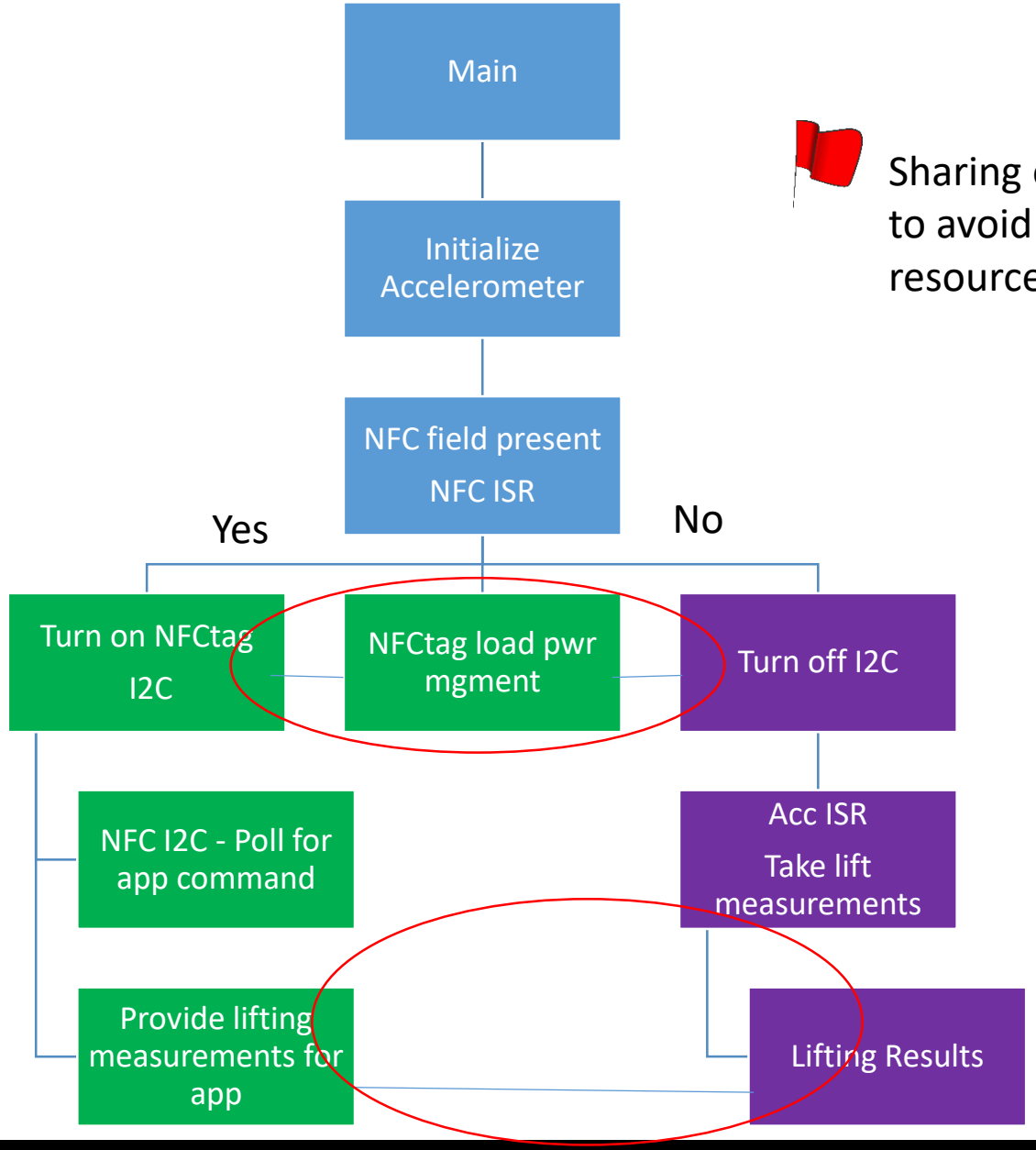


Lifting Accelerometer Software Block Diagram

Firmware best practices

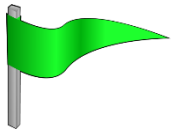


Organizational Chart

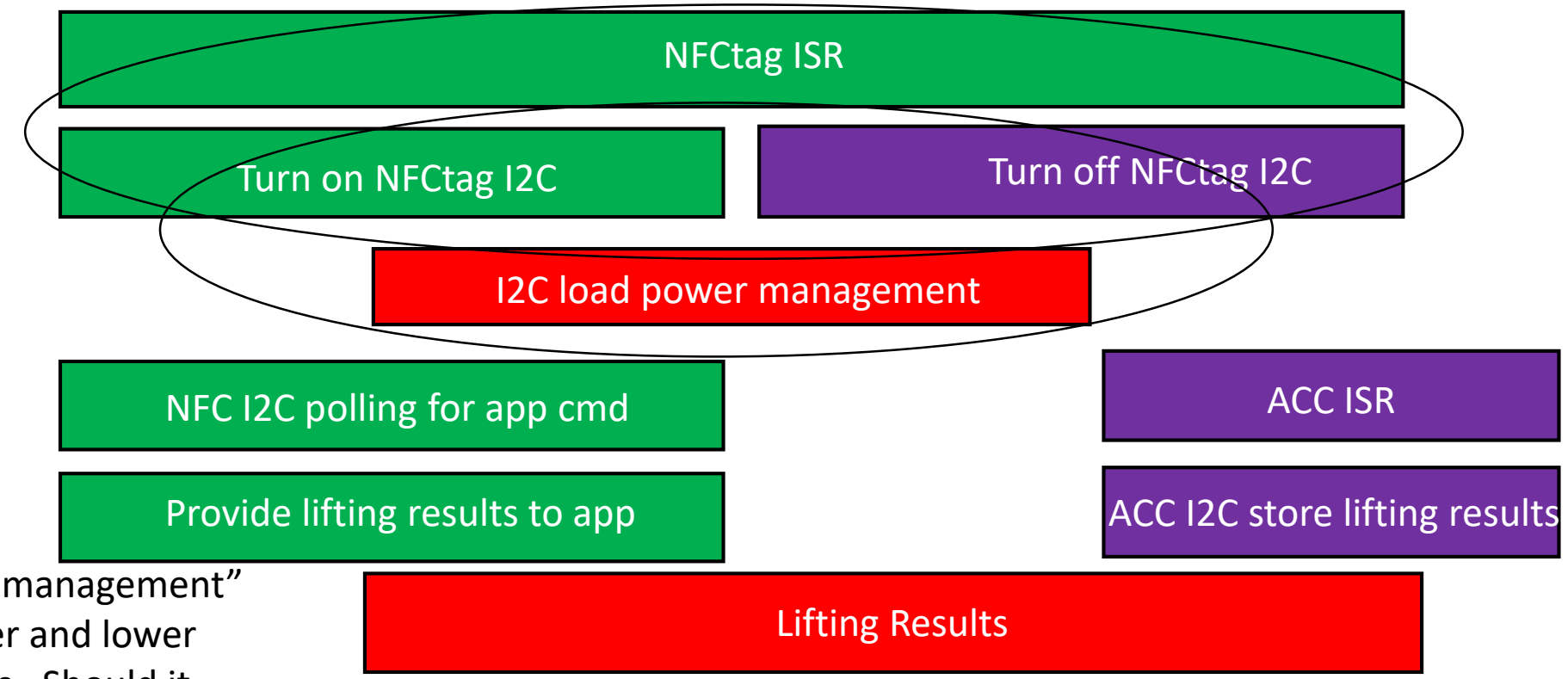


Sharing of a resource is a red flag to avoid contention around the resource

Firmware best practices

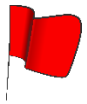


"Turn on NFCtag I2C" and "Turn off NFCtag I2C share both the top and lower levels. Should they merge into one module?

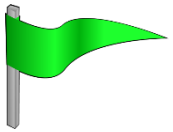


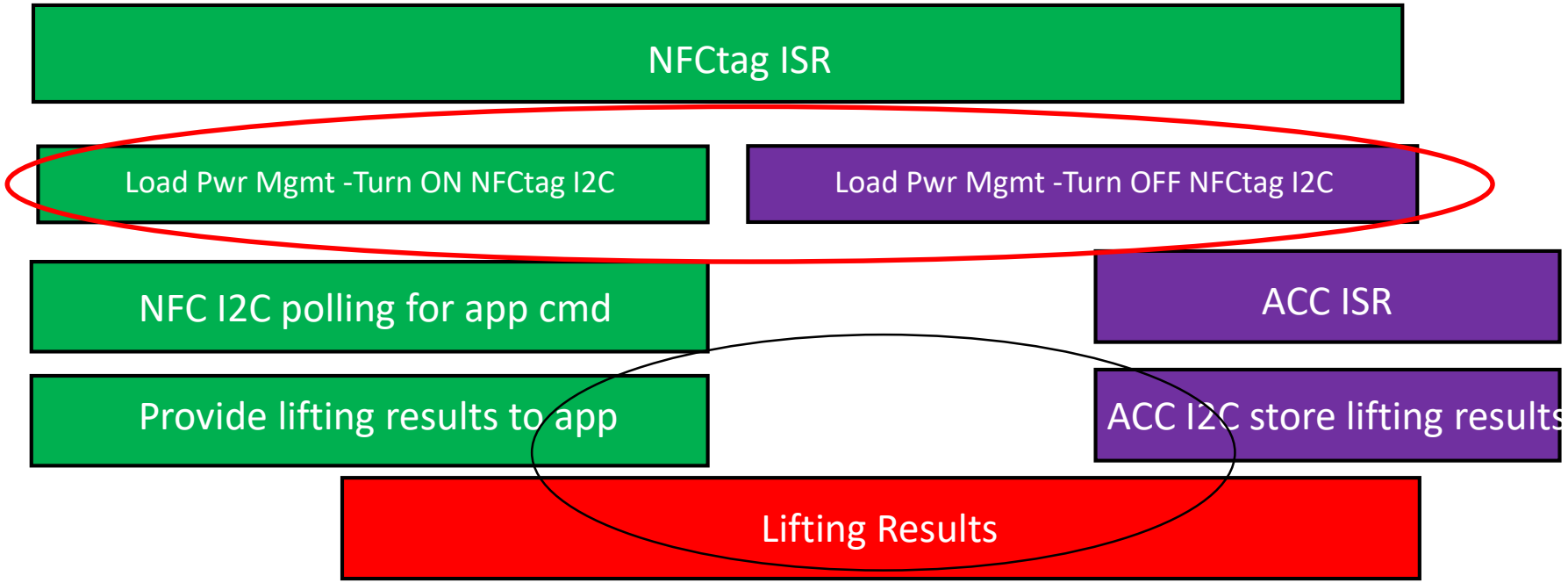
"I2C load power management" share both higher and lower level connections. Should it merge with another module?

Firmware best practices



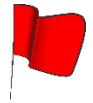
Sharing of a resource is a red flag to avoid contention around the resource





"Making Embedded Systems," by Elecia White, ISBN: 978-1-449-30214-6

Firmware best practices

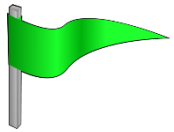
- Why is a shared resource a  ?
  - Communication between more two or more tasks or modules using a shared resource usually is through a global variable
  - In an event based system, the setting and clearing of global variables can be interrupt and changing the flow of the task
  - The uncertainty of the timing to update or read a global variable can be result in data race condition
    - Task 1 will read the global variable and modify it
      - Read global variable (button pressed)
      - Processes the global variable request
      - Clears global variable (button request cleared)

But, task 2 interrupts tasks 1 sets button pressed

Will the system actually see the 2nd button request? Data race condition

Firmware best practices

- To avoid data race conditions, reading and writing to shared memory, global variables, multiple tasks must be prohibited from accessing this memory with uncertainty
  - Any time memory is shared between tasks is read or written, it creates a critical section of code
    - Access to a shared resource
  - The shared resource must be protected so only one task can modify it at a time
    - Process is called mutual exclusion, shorten to mutex



  - To insure that the task's access to the shared resource is protected, the operation must be atomic

Firmware best practices

This instruction breaks down to a read modify write:
1. Read i
2. add 1 to the loaded value of i
3. write i back

Multiple instructions cannot be atomic since an interrupt can occur between any instruction

- Atomic action
  - An operation that cannot be interrupted by anything else in the system

- Which of these operation is atomic?
  - i++;
  - ADC0 -> IEN |= ADC_IEN_SINGLE;;
  - Subs R1, 1
  - For (j = 0; j<1000; j++);

This instruction breaks down to a read modify write:
1. Read ADC0->IEN
2. Or loaded ADC0->IEN with ADC_IEN_SINGLE
3. Write ADC0->IEN

Multiple instructions cannot be atomic since an interrupt can occur between any instruction