

ECEN 5823-001 / -001B

Internet of Things Embedded Firmware

Lecture #14

12 October 2017

Agenda

- Class announcements
- Setting up a BLE Service
- Bluetooth Smart

Class Announcements

- Quiz #7 is due at 11:59pm on Sunday, October 15th, 2017
- BLE Health Temperature Service assignment is due at 11:59pm on Sunday, October 22nd, 2017
- Mid-term will be held in class on **Tuesday, 24th, at 6:30pm in class**
 - For on campus students, you must be in class for the exam
 - For distant learners, the mid-term will be due by 11:59pm on **Thursday, October 26th, 2017**
- There will be no homework assignment or quiz the week of October 16th

Mid-Term

- Tuesday, 24th, at 6:30pm in class
 - For on campus students, you must be in class for the exam
 - For distant learners, the mid-term will be due by 11:59pm on Thursday, October 26th, 2017
- Will be administered by D2L
 - 75 minute time limit for the Mid-term
 - 5 minutes time limit for the bonus section
 - 1 attempt
- Open book, but **not** open people, **not** google

Mid-Term

- Material covered will include:
 - All the readings from the first day of class
 - All the lectures through **Thursday, October 19th**, 2017
 - All assignments
- Questions:
 - 33 questions that will represent 100% of the mid-term
 - Question pool will be over 100 questions
 - 10 bonus questions each worth 1 point
 - Comprised of a random selection from the first 7 week quiz questions (roughly 150 questions in the question library)

Using the Bluetooth Stack on the Blue Gecko

C Functions

```
/* Event id */
gecko_evt_gatt_server_characteristic_status_id

/* Event structure */
struct gecko_msg_gatt_server_characteristic_status_evt_t
{
    uint8 connection;,
    uint16 characteristic;,
    uint8 status_flags;,
    uint16 client_config_flags;
};
```

characterist_status = **evt->data.gatt_server_characterist_status.status_flags;**

Using the Bluetooth Stack on the Blue Gecko

```
While (1) {  
    evt = gecko_wait_event();  
    switch (BGLIB_MSG_ID(evt->header)){  
        case gecko_evt_system_boot_id:  
            Break;  
        case gecko_evt_le_connection_closed_id:  
            Break;  
        default:  
            Break;  
    }  
}
```

Using the Bluetooth Stack on the Blue Gecko

- case `gecko_evt_system_boot_id`:
 - Used for the first time to initialize the Bluetooth stack and specify the type of advertising

```
/* Set advertising parameters. The third parameter '7' sets  
advertising on all channels. */
```

```
    gecko_cmd_le_gap_set_adv_parameters(Bt_Min_Adv,  
Bt_Max_Adv, BT_Adv_Channels);
```

```
/* Start general advertising and enable connections. */
```

```
    gecko_cmd_le_gap_set_mode(le_gap_general_discoverable,  
le_gap_undirected_connectable);
```

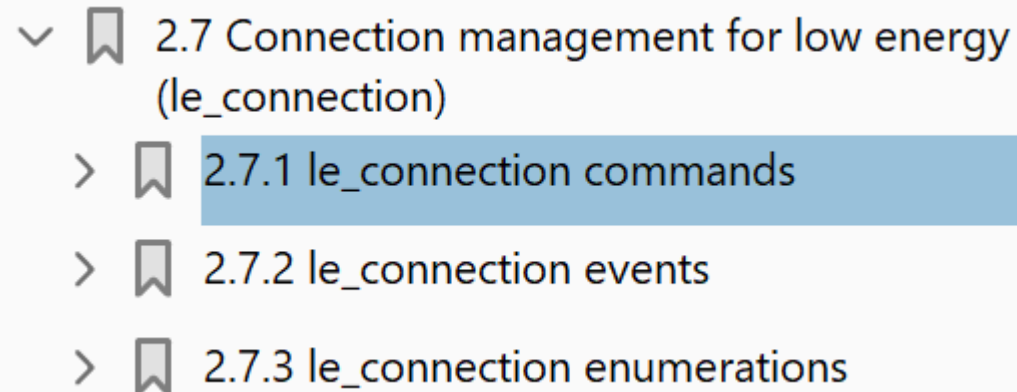

Using the Bluetooth Stack on the Blue Gecko

- case gecko_evt_le_connection_closed_id:
 - Used to reset the Bluetooth device to begin advertising
- **case gecko_evt_le_connection_closed_id:**

```
/* Check if need to boot to dfu mode */
if (boot_to_dfu) {
    /* Enter to DFU OTA mode */
    gecko_cmd_system_reset(2);
} else {
    /* Restart advertising after client has disconnected */
    gecko_cmd_le_gap_set_mode(le_gap_general_discoverable,
le_gap_undirected_connectable);
}
break;
```

Using the Bluetooth Stack on the Blue Gecko

- The Bluetooth stacks all specifies commands, cmd, and responses
- Responses can be read back immediately after issuing the command



Using the Bluetooth Stack on the Blue Gecko

- Command:
 - `gecko_cmd_le_connection_disable_slave_latency(uint8 connection, uint8 disable);`
- Response:
 - `evt->data.rsp_le_connection_disable_slave_latency.result`

```
/* Function */
struct gecko_msg_le_connection_disable_slave_latency_rsp_t *gecko_cmd_le_connection_disable_slave_latency(uint8 connection, uint8 disable);

/* Response id */
gecko_rsp_le_connection_disable_slave_latency_id

/* Response structure */
struct gecko_msg_le_connection_disable_slave_latency_rsp_t
{
    uint16 result;
};
```

Using the Bluetooth Stack on the Blue Gecko

- You can send a command to send a notification and indication by using the appropriate Bluetooth Stack API
- You will be using this command to send the temperature via the Health Temperature Service

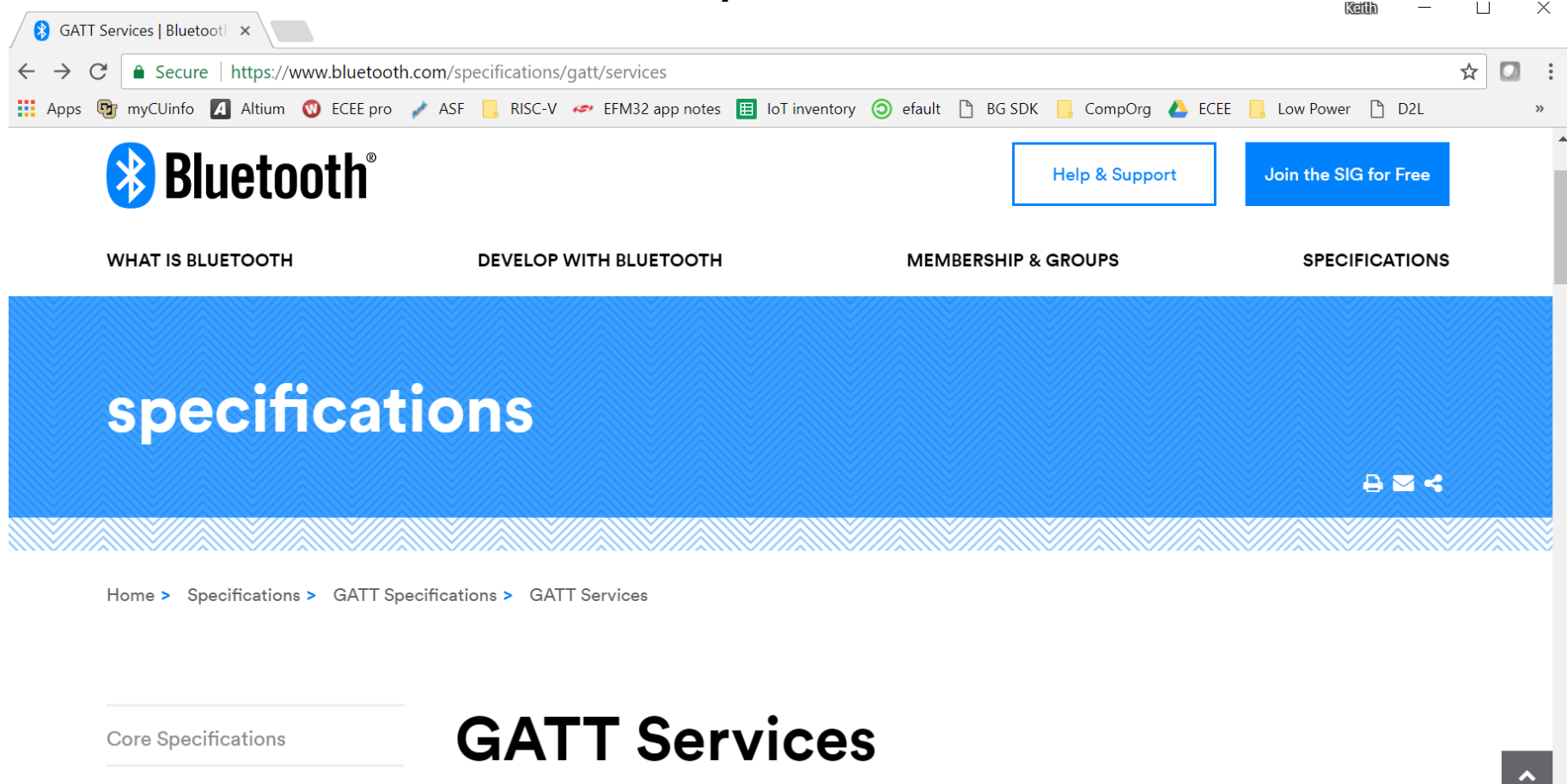
2.5.1.4 cmd_gatt_server_send_characteristic_notification

This command can be used to send notifications or indications to a remote GATT client. At most ATT_MTU - 3 amount of data can be sent once. Notification or indication is sent only if the client has enabled them by setting the corresponding flag to the Client Characteristic Configuration descriptor. A new indication cannot be sent before a confirmation from the GATT client is first received. The confirmation is indicated by [gatt_server_characteristic_status event](#).

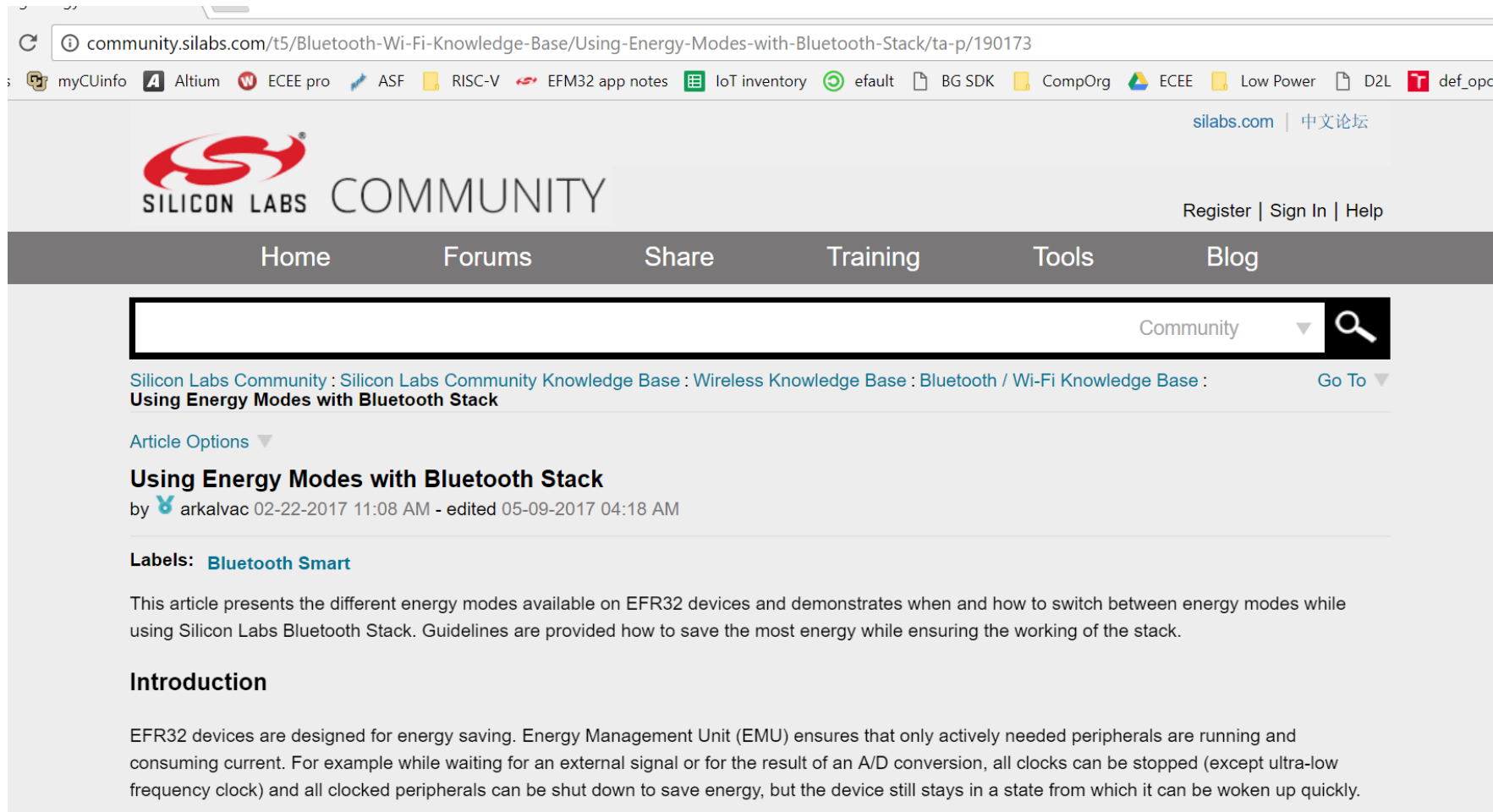
Table 2.111. Command

Byte	Type	Name	Description
0	0x20	hlen	Message type: Command
1	0x04	lolen	Minimum payload length
2	0x0a	class	Message class: Generic Attribute Profile Server
3	0x05	method	Message ID
4	uint8	connection	Handle of the connection over which the notification or indication is sent. Values: <ul style="list-style-type: none"> • 0xff: Sends notification or indication to all connected devices. • Other: Connection handle
5-6	uint16	characteristic	Characteristic handle
7	uint8array	value	Value to be notified or indicated

Bluetooth website to further understand defined services and profiles



Setting Application Energy Modes



The screenshot shows a web browser displaying the Silicon Labs Community forum. The URL in the address bar is community.silabs.com/t5/Bluetooth-Wi-Fi-Knowledge-Base/Using-Energy-Modes-with-Bluetooth-Stack/ta-p/190173. The page features a navigation bar with links for Home, Forums, Share, Training, Tools, and Blog. A search bar is located below the navigation bar. The main content area displays the title 'Using Energy Modes with Bluetooth Stack' by user 'arkalvac', dated 02-22-2017. The article is labeled 'Bluetooth Smart' and includes an introduction section discussing energy saving on EFR32 devices.

community.silabs.com/t5/Bluetooth-Wi-Fi-Knowledge-Base/Using-Energy-Modes-with-Bluetooth-Stack/ta-p/190173

myCUinfo Altium ECEE pro ASF RISC-V EFM32 app notes IoT inventory efault BG SDK CompOrg ECEE Low Power D2L def_opc

silabs.com | 中文论坛

SILICON LABS COMMUNITY

Register | Sign In | Help

Home Forums Share Training Tools Blog

Community

Silicon Labs Community : Silicon Labs Community Knowledge Base : Wireless Knowledge Base : Bluetooth / Wi-Fi Knowledge Base : [Go To](#)

Using Energy Modes with Bluetooth Stack

by [arkalvac](#) 02-22-2017 11:08 AM - edited 05-09-2017 04:18 AM

Labels: [Bluetooth Smart](#)

This article presents the different energy modes available on EFR32 devices and demonstrates when and how to switch between energy modes while using Silicon Labs Bluetooth Stack. Guidelines are provided how to save the most energy while ensuring the working of the stack.

Introduction

EFR32 devices are designed for energy saving. Energy Management Unit (EMU) ensures that only actively needed peripherals are running and consuming current. For example while waiting for an external signal or for the result of an A/D conversion, all clocks can be stopped (except ultra-low frequency clock) and all clocked peripherals can be shut down to save energy, but the device still stays in a state from which it can be woken up quickly.

Setting Application Energy Modes

Enabling EM2 Deep Sleep in the Stack

Regarding sleep policy Silicon Labs Bluetooth stack can be configured in two ways: putting the device into EM1 Sleep mode or into EM2 Deep Sleep mode, while EM0 Active mode is not needed. If sleep mode is not configured, EM1 is used. To enable EM2 Deep Sleep in BGScript add this line to the hardware.xml configuration file:

```
<sleep enable="true" />
```

To enable Deep Sleep in a C project, add this line to the config structure, which will be passed to `gecko_init()`:

```
config.sleep.flags=SLEEP_FLAGS_DEEP_SLEEP_ENABLE;
```

In the software examples provided with Silicon Labs Bluetooth SDK Deep Sleep is enabled by default (except NCP empty target).

The stack puts the device into EM1 Sleep or into EM2 Deep Sleep mode every time the processor is not needed, and starts a timer to wake it up when needed: when a communication window opens or a task is to be done.

The default template of a Bluetooth stack based application looks like this

```
while(1)
{
    evt = gecko_wait_event();

    switch(BGLIB_MSG_ID(evt->header))
    {
        //handling of different stack events
    }
}
```

Setting Application Energy Modes

Temporarily Disable EM2 Deep Sleep Mode

The Deep Sleep enable bit in the stack configuration cannot be dynamically changed. That is once Deep Sleep is enabled/disabled, it cannot be withdrawn. However, there are two ways to temporarily disable going to EM2 Deep Sleep mode: using sleep driver or using wake up pin.

With sleep driver the EM2 Deep sleep mode can be disabled (/blocked) temporarily by using **SLEEP_SleepBlockBegin(sleepEM2)**. To Re-enable EM2 Deep Sleep mode use **SLEEP_SleepBlockEnd(sleepEM2)**. If EM2 is disabled (/blocked), then the stack will switch between EM0 and EM1 temporarily.

To access these functions sleep.h has to be included in your source file! sleep.c is already precompiled with the stack, hence it has not to be added to the project!

Note, that multiple issuing of SLEEP_SleepBlockBegin() requires multiple issuing of SLEEP_Sleep_BlockEnd(). Every SLEEP_SleepBlockBegin() increases the corresponding counter and every SLEEP_SleepBlockEnd() decreases it.

Setting Application Energy Modes



UG136: Silicon Labs *Bluetooth*® C Application Developer's Guide



This document is an essential reference for everybody developing C-based applications for the Silicon Labs Wireless Gecko products using the Silicon Labs Bluetooth stack. The guide covers the Bluetooth stack architecture, application development flow, usage and limitations of the MCU core and peripherals, stack configuration options, and stack resource usage.

The purpose of the document is to capture and fill in the blanks between the Bluetooth Stack API reference, Gecko SDK API reference, and Wireless Gecko reference manuals, when developing Bluetooth applications for the Wireless Geckos. This document exposes details that will help developers make the most out of the available hardware resources.

KEY POINTS

- Project structure and development flow
- Bluetooth stack and Wireless Gecko configuration
- Interrupt handling
- Event and sleep management
- Resource usage and available resources

```
/**
 * Main
 */
void main()
{
    ...

    //Event loop
    while(1)
    {
        ...

        //External signal indication (comes from the interrupt handler)
        case gecko_evt_system_external_signal_id:
            // Handle GPIO IRQ and do something
            // External signal command's parameter can be accessed using
            // event->data.evt_system_external_signal.extsignals
            break;
        ...
    }
}

/**
 * Handle GPIO interrupts and trigger system_external_signal event
 */
void GPIO_ODD_IRQHandler()
{
    static bool radioHalted = false;

    uint32_t flags = GPIO_IntGet();
    GPIO_IntClear(flags);

    //Send gecko_evt_system_external_signal_id event to the main loop
    gecko_external_signal(...);
}
}
```

For now ...

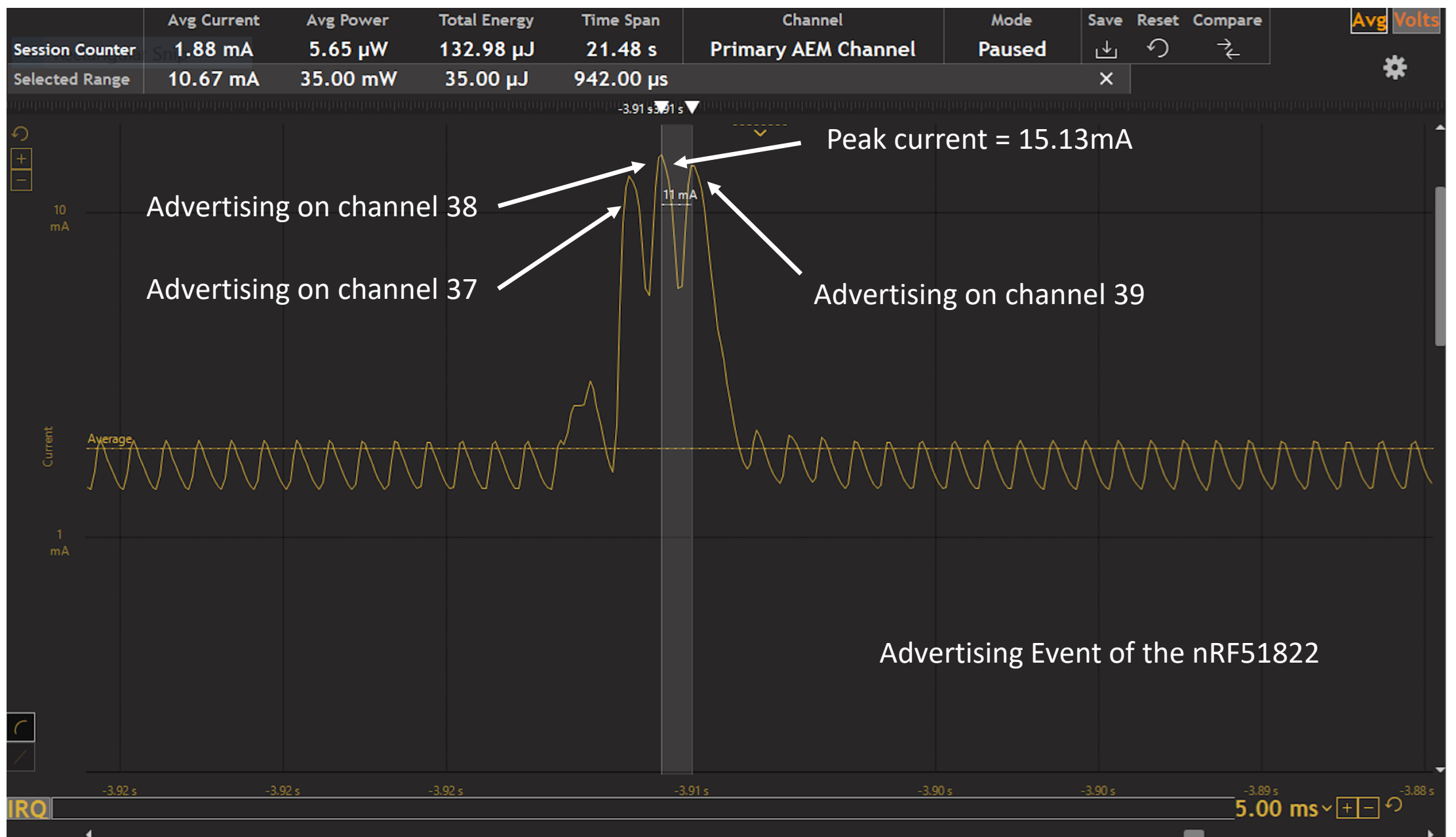
- **Disable** the application Enter Sleep Routines
- I will be learning/experimenting more this weekend and I will update the class on Tuesday

BLE: Peripherals (Discoverable Advertising)

- One of the fundamental ways to optimize for low power is to appropriately choose the intervals for advertising and connection intervals
 - The choice could be the difference between a few weeks of operation to years
- Typically, the first state that a peripheral performs is discoverable advertising so that a central device can find it
 - The time that a peripheral is in this state of operation is typically very short in the lifetime of the device since in most applications, the user will want to connect a device shortly after installation
 - Once bonded to a central device, it will move into connectable advertising

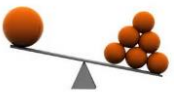
BLE: Peripherals (Discoverable Advertising)

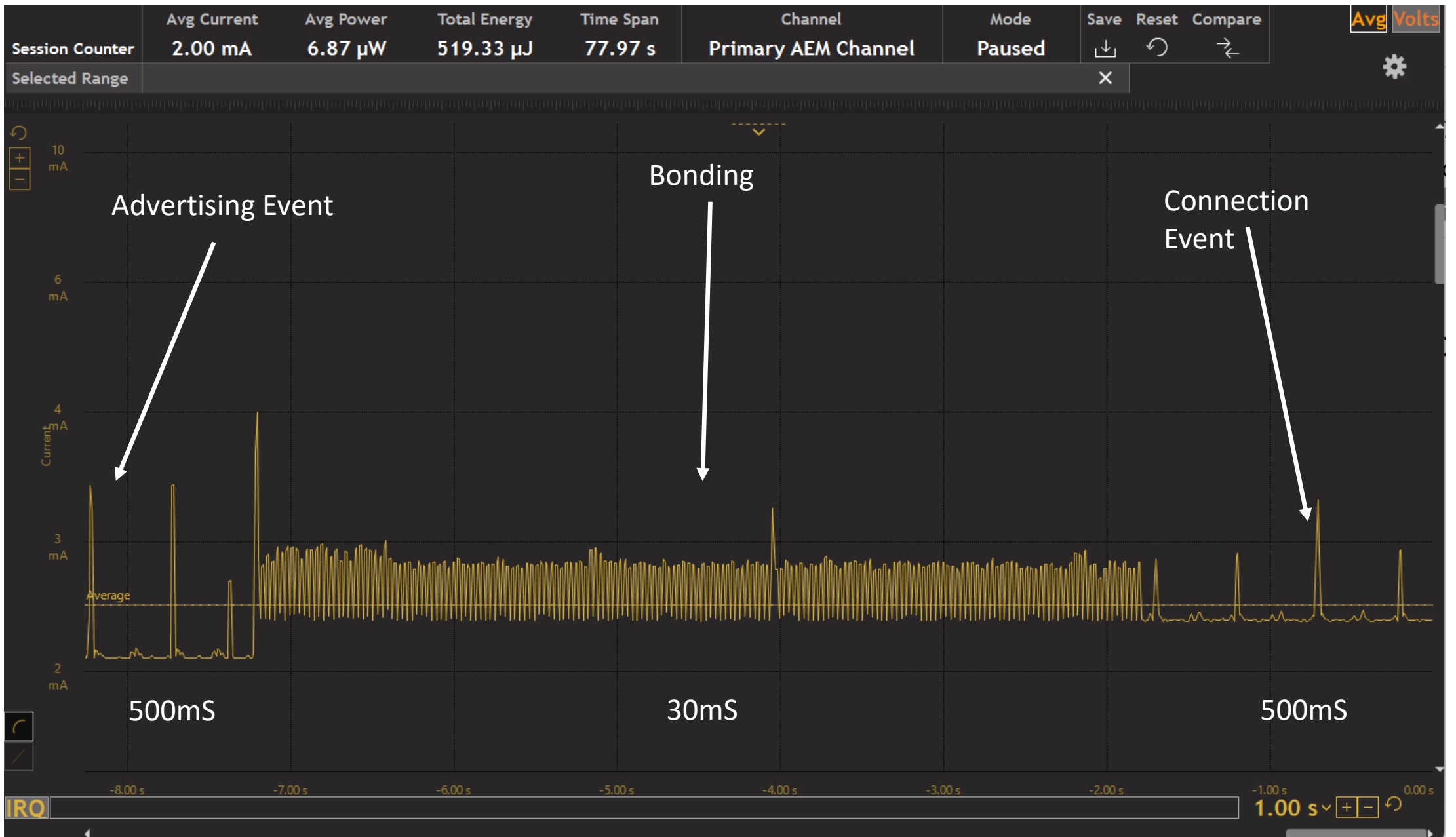
- With the idea that the peripheral will be in the discoverable advertising state for a very short period of its life cycle and to provide a good user experience of a quick connect, the advertising interval of 250mS would be a good compromise between speed of discovery and power savings



BLE: Peripherals (Bonding)

- After the peripheral makes a connection from the discoverable advertising state, the device enters bonding and the connection interval can be set very short (or fast)
 - A fast connection interval of 7.5mS to 25mS can use a lot of power, but it also allows the central device to discover the set of services and characteristics that the peripheral has to offer as well as provide prompt feedback to the user about how it can interact with the peripheral
 - If the connection interval is very slow, 1 to 4S, it can be an extremely long time before a central device can determine how to utilize the peripheral
- Once the appropriate bonding information has been exchanged, the connection interval timing can be long (or slow) to conserve energy





BLE: Peripherals (Connectable Advertising)

- After a peripheral has bonded with a central device, it can disconnect and at a later time advertise to allow the central device to reconnect
- The advertising interval used in this state is a compromise between how fast a central device can reconnect to the peripheral and the power consumption that the peripheral will use when disconnected
 - Example: A heart-rate belt used by a runner might only be connected for the 3 hours a week while the runner is using it while the remaining 165 hours a week it remains in the connectable advertising state. It also does not need to connect instantaneously, so using a longer connectable advertising interval would be advisable or even disconnect when the belt is not worn
 - A connectable advertising interval of 1 second would allow a central device to be able to connect within a few seconds, and if the peripheral requires a faster connection time, than a connectable advertising interval of 0.5s or less would be required

BLE: Peripheral (Directed Advertising)

- Directed Advertising is used when a peripheral needs to connect directly to a central device usually based on an event happening
- And, when the time between the event and sending notification to the central device must be as short as possible
 - Example: A fire or smoke alarm
- A peripheral burns a lot of power/energy when executing Directed Advertising because the peripheral transmit lots of advertising packets very quickly to a single central device
 - If the central device is available to initiate a connection to this peripheral, it will immediately connect, allowing the peripheral to send its data quickly

BLE: Peripheral (Directed Advertising)

- Directed advertising is the quickest way for a peripheral to make a connection to a central device
 - Connection times + sending the required data can be less than 3 milliseconds
- There is no interval configuration in Directed Advertising
 - Every 3.75mS, the peripheral will send its advertising packets on each of the 3 advertising channels
 - Resulting in one directed advertising packet every 1.25mS or 800 packets per second
 - If there is no central device to respond, directed advertising could consume 250+ times more energy per second compared to a device sending connectable advertising packets once per second
- Directed Advertising could be very useful for peripherals that rarely need to be connected, but when the need arises, connected very quickly

BLE: Peripheral (Connected)

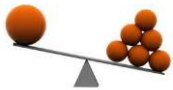
- When connected, the central device has complete control over the connection intervals and latency used by the peripheral
- The peripheral does not have a way to signal to the central device that the current connection values being used are appropriate
 - But, the peripheral can request or suggest alternative settings after connection has been made
- There are two configurable values for the connection parameters that relate to power consumption
 - **Connection Interval** (`connInterval`): The time that determines how often the central will transmit and synchronize with the peripheral. The `connInterval` is a multiple of 1.25ms
 - **Slave Latency** (`slavelatency`): The number of master connection intervals that the slave can ignore. This value can be 0 to 500

BLE: Peripheral (Connection)

- Example 1:
 - connInterval is set to 12.5mS (10 times 1.25mS)
 - slaveLatency is set to 0
 - This would result in the slave required to listen every 12.5mS to determine if the master has a request and consuming a lot of energy
- Example 2:
 - connInterval is set to 12.5mS (10 times 1.25mS)
 - slaveLatency is set to 1
 - This would enable the slave to skip one of the reserved connection events for the peripheral, but must listen to the second connection event
 - This would result in a 50% power reduction due to halving the time that the peripheral listens and transmits data to the central device
 - The peripheral could listen and transmit to the first 12.5mS connection event if required

BLE: Peripheral (Connection)

- **IMPORTANT:** Slave Latency also sets the latency of the central device to the peripheral
 - For example: If a keyboard has an LED indicator that needs to be turned on within 500mS for the desired user experience, the `connInterval` X `slavelatency` must be less than 500mS
- The slave has the option of jumping onto a connection event early, but the master does not!



BLE: Peripheral (Connection)

- **IMPORTANT:** Extremely long Slave Latency can actually consume more energy than they actually save due to the accuracy of the central and peripheral clocks
 - In worst case, the clock inaccuracies could be 500 parts per million on each device, and possibly in opposition directions
 - If a device was set with a connection interval of 15mS and a slave latency of 500, the slave only would be required to listen every 7.5s
 - At the end of 7.5s, the two devices could be out of synch by 3.75mS and possibly up to 7.5mS if the two devices are off in different directions
 - To resolve this possible timing synchronization issue, the peripheral would need to begin to listen 7.5mS before its expected connection time and possibly 7.5mS afterwards . This is called [window widening](#).
 - For every 7.5s, the peripheral would have to listen from 7.5 to 15.0mS to synchronize
 - Resulting in significant energy loss

BLE: Peripheral (Connection)

- For practical purposes in terms of saving energy, it does not make sense to set the slave latency to have a maximum time between connections greater than 1s or fewer than 300mS
 - Below 300mS, the power used to repeatedly synchronize is higher than it would be to wait long
 - Above 1s, the power used by window widening does not save any significant amount of power, and the user experience is enhanced with a smaller slave latency