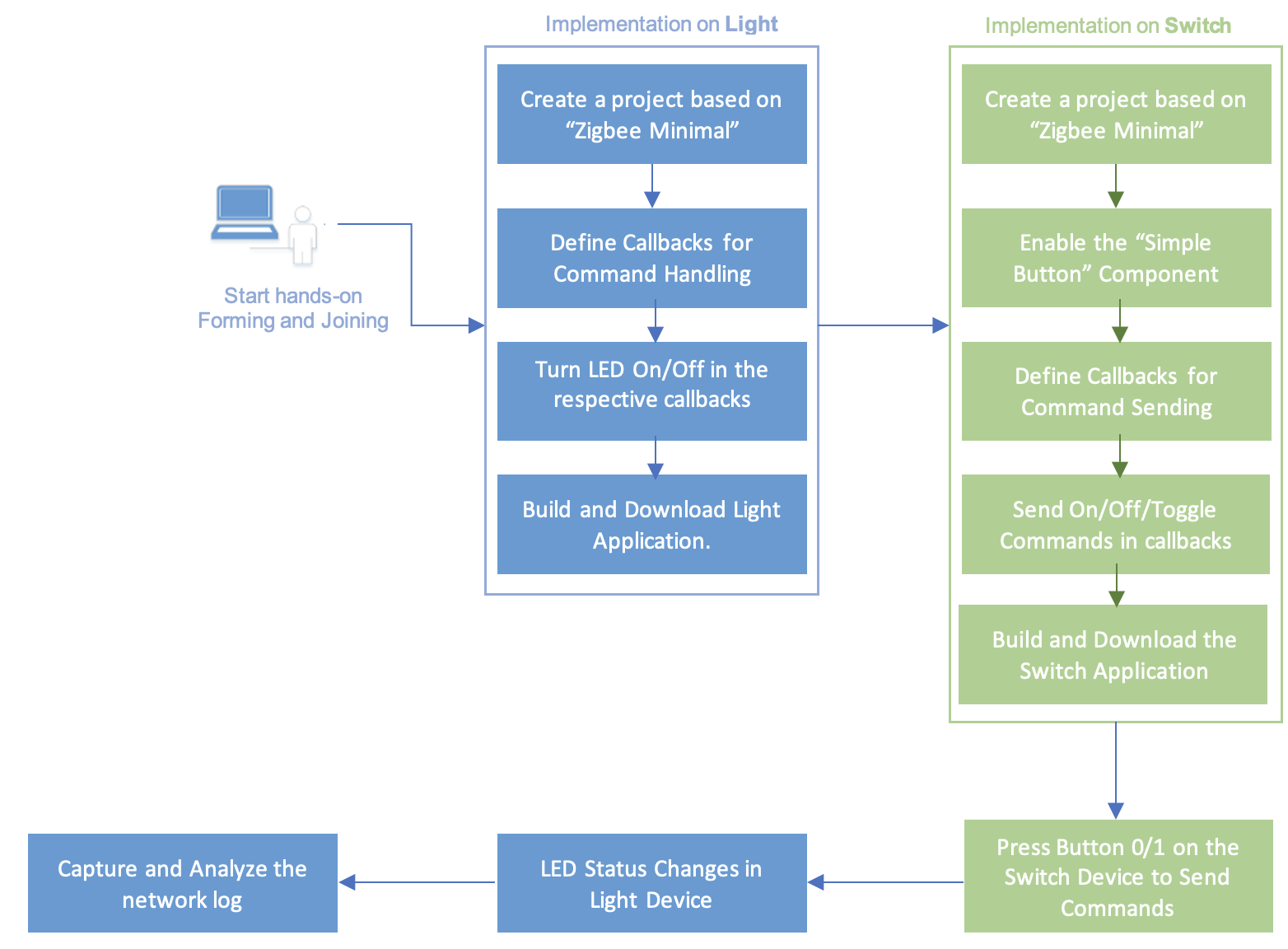
Original Source: <https://github.com/SiliconLabs/zigbee_applications/blob/f7059cdff81e6c57b711aaa2f0bd3634e2300b6f/zigbee_bootcamp/Zigbee-Hands-on-Sending-OnOff-Commands/Zigbee-Hands-on-Sending-OnOff-Commands.md>  
  
  
**Sending On/Off Commands**

From the FormingAndJoiningZigbee document, we created two projects, Zigbee\_Light\_ZC and Zigbee\_Switch\_ZR. These have been programmed into two separate devices which are now in the same network and ready to transmit and receive data on the network. In this module, the Switch device should send one of the On/Off commands based on which button has been pressed. The Light application should respond by turning the LED1 on/off based on the received command. Our task is to prepare the devices for these features.

In this module, we will demonstrate how to send ZCL on-off commands from the Switch node to operate the LED in the Light node. As in the previous hands-on, the network will consist of two devices using BRD4162A (EFR32MG12). The figure below depicts this hands-on workflow:  
  


**Command Handling on the Light Device:**

To become aware of any received command from the user application level, the emberAfPostAttributeChangeCallback callback function will be used. This callback is defined inside of the "WEAK" macro in protocol/zigbee/app/framework/util/global-callback.h. The application can override each of these weak definitions by simply adding a callback implementation to some application-level source file in this case we will do so in **app.c.**

Implement the application code as below:

Add these headers for accessing led instances:  
  
#include "app/framework/include/af.h"

#include "sl\_simple\_led\_instances.h"

#include "sl\_simple\_led.h"

Now add this function without removing the autogenerated code which created with Zigbee Minimal Configuration:

void emberAfPostAttributeChangeCallback(uint8\_t endpoint,

EmberAfClusterId clusterId,

EmberAfAttributeId attributeId,

uint8\_t mask,

uint16\_t manufacturerCode,

uint8\_t type,

uint8\_t size,

uint8\_t\* value)

{

if (clusterId == ZCL\_ON\_OFF\_CLUSTER\_ID

&& attributeId == ZCL\_ON\_OFF\_ATTRIBUTE\_ID

&& mask == CLUSTER\_MASK\_SERVER) {

bool onOff;

if (emberAfReadServerAttribute(endpoint,

ZCL\_ON\_OFF\_CLUSTER\_ID,

ZCL\_ON\_OFF\_ATTRIBUTE\_ID,

(uint8\_t \*)&onOff,

sizeof(onOff))

== EMBER\_ZCL\_STATUS\_SUCCESS) {

if (onOff) {

sl\_led\_turn\_on(&sl\_led\_led0);

} else {

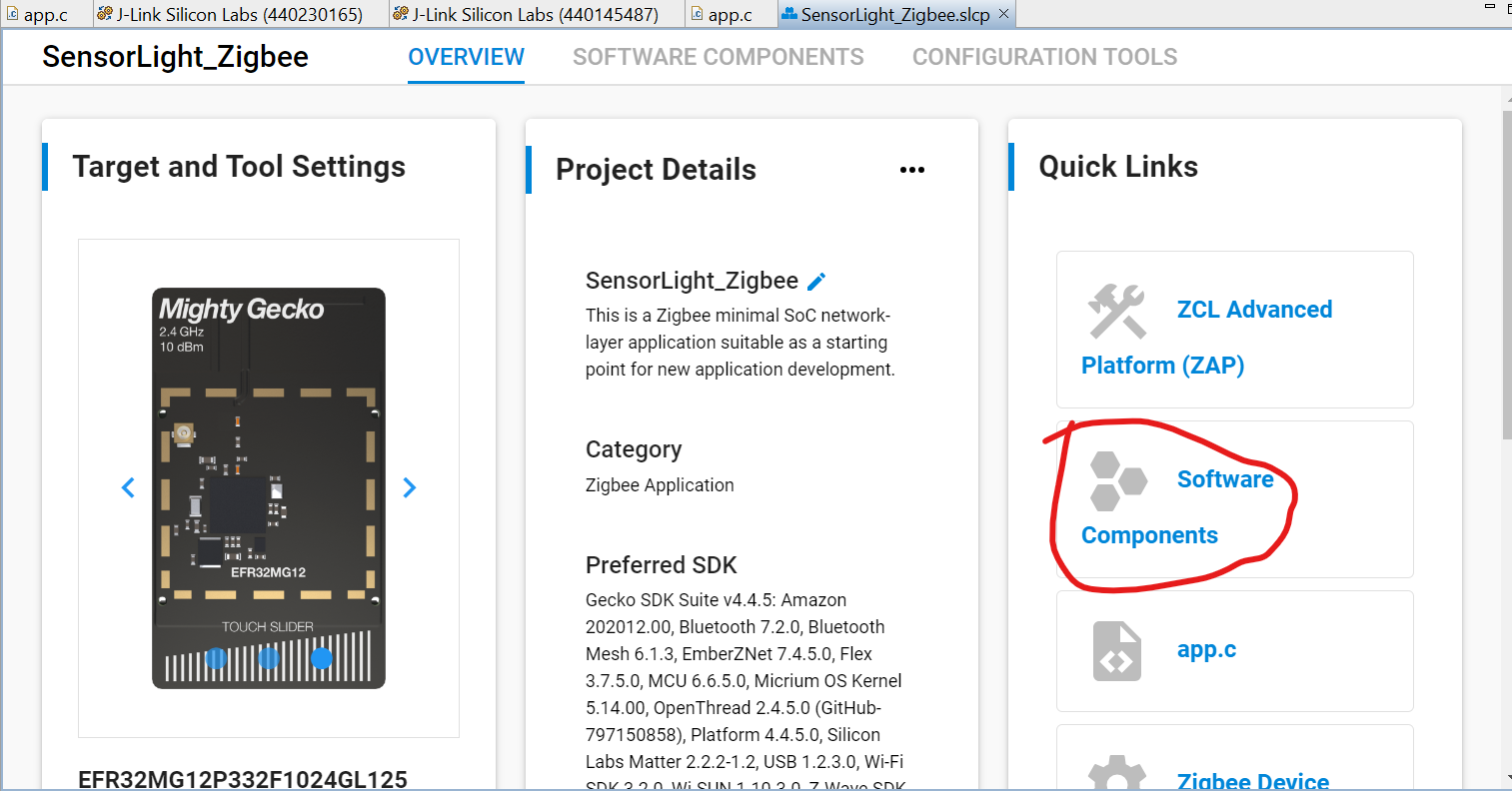
sl\_led\_turn\_off(&sl\_led\_led0);

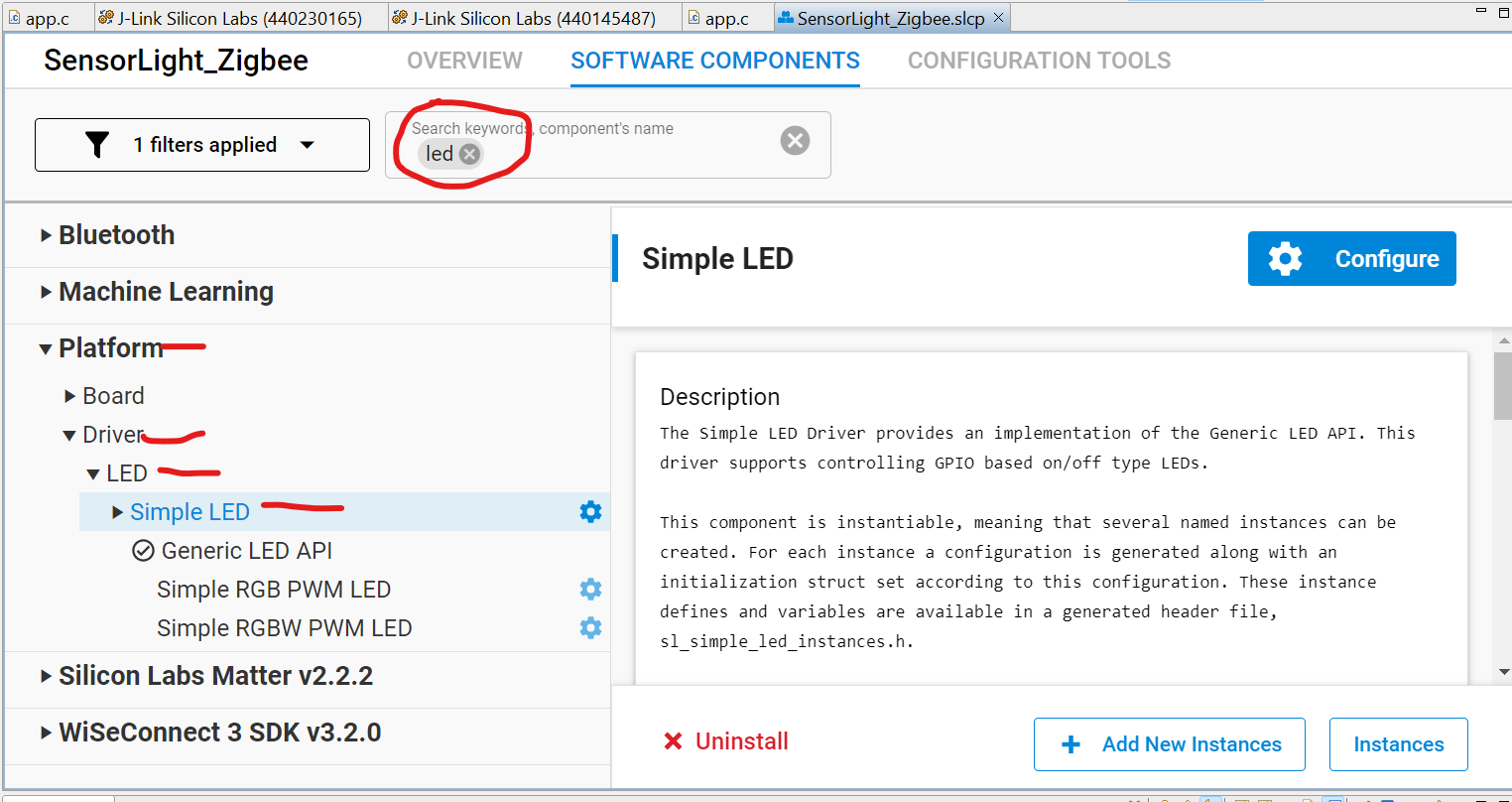
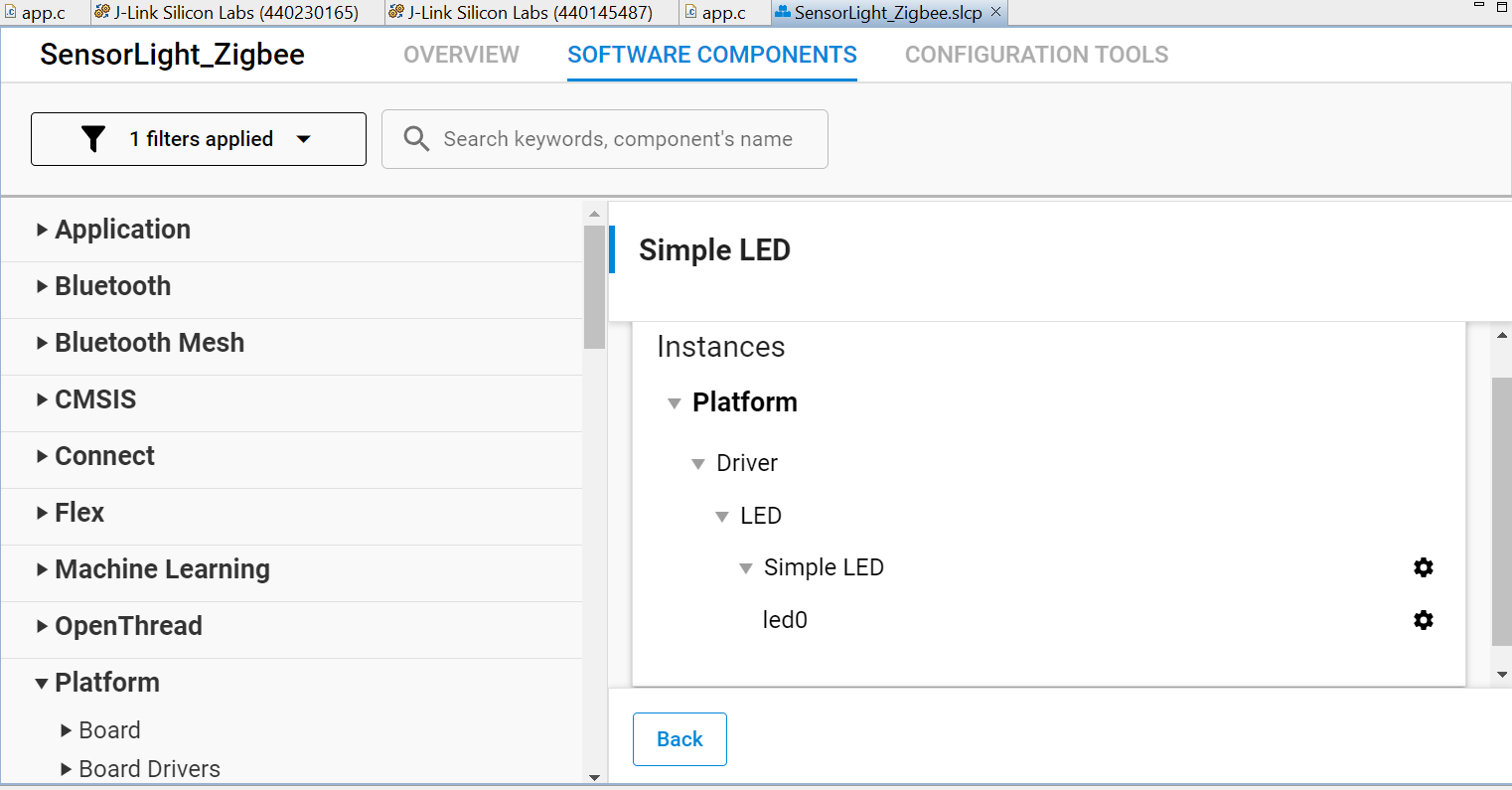
}

}

}

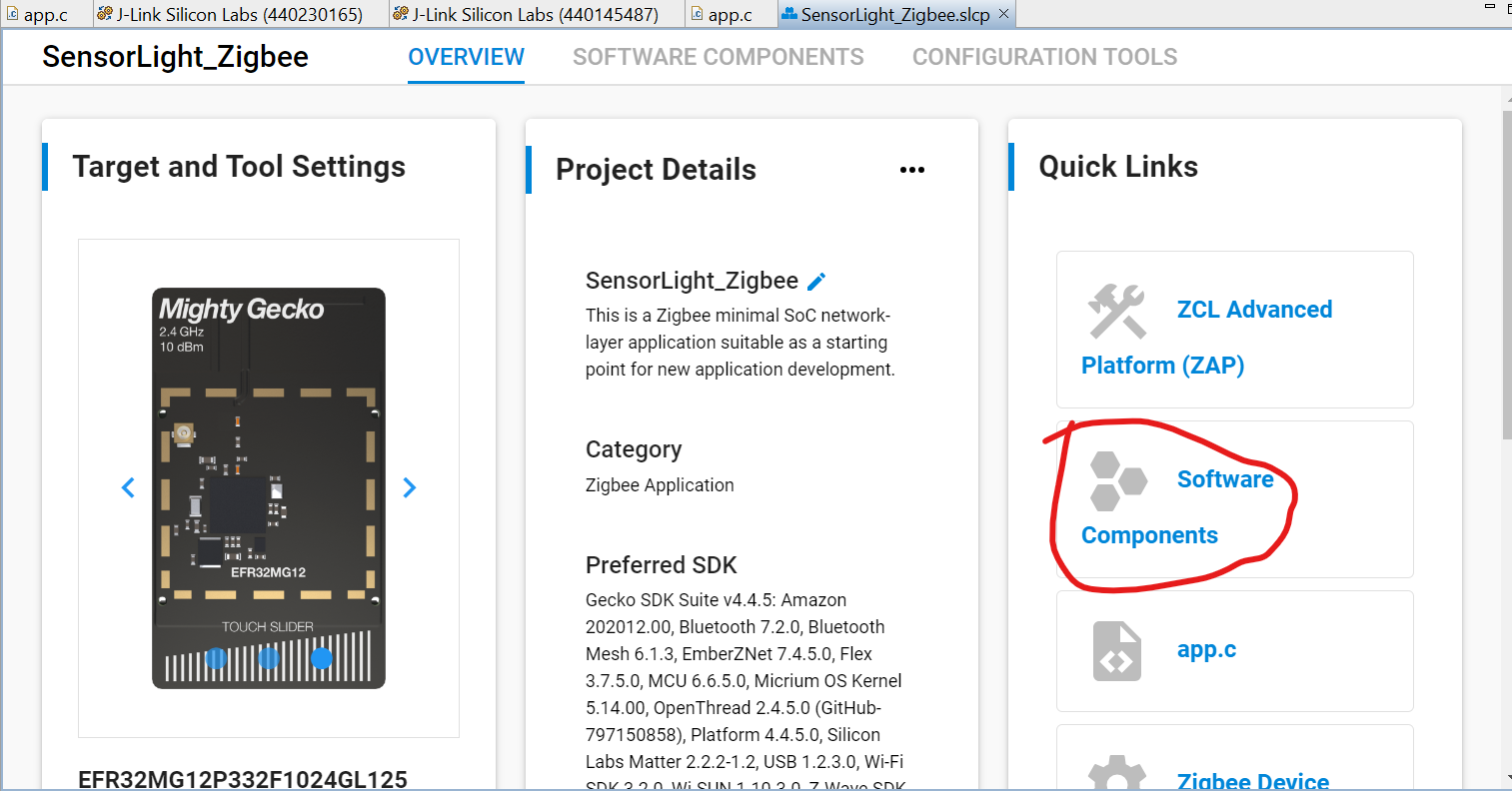
}

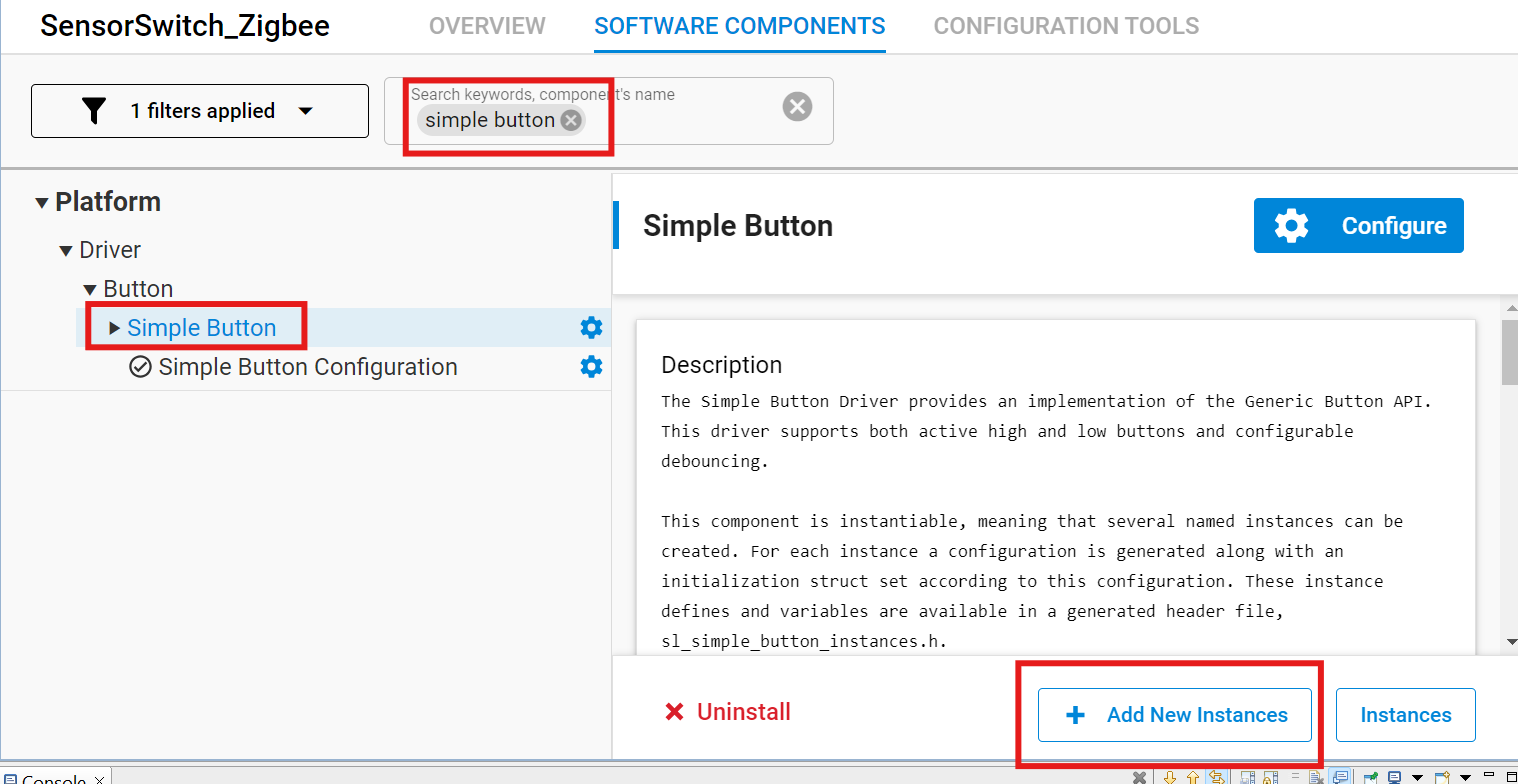
Note: Led instances are already created with the Zigbee Minimal Configuration. If you want to know how to create the led instances follow Section Led Instances Creation. As we already have the led instances created so we can ignore them for now.  
  
**LED Instances Creation:**  
1) Navigate to Software Components :   
  


2) Search For Led and follow the below picture for navigation :   
  
  
  
3) As It is already created we can see instances of LED which are created by navigating to Instances, Here by navigating to Add New Instances we can create more instances:   


Now Lets Continue With Normal Integration of the Code:

**Command Sending from Switch Device.**

Moving to the Switch device application, the objective is to make use of the two WSTK buttons (BTN0 and BTN1), to turn the Light LED ON and OFF. The button operations are handled by the Platform > Driver > Button > Simple Button component, this component must be installed and two different instances btn0 and btn1 should be created as depicted in below figures.  
  


  
  
You can find the Simple Button component defined callbacks [here](https://docs.silabs.com/gecko-platform/4.1/driver/api/group-simple-button). Once the button changes its state void sl\_button\_on\_change(const sl\_button\_t \*handle) is called in an interrupt context. Similar to **Command Handling on Light Device** we will implement this callback in the **app.c** source file. In order to send commands it is important to consider that every command is stored in a buffer before being sent. The transmitted data buffer should be built up as follows:

The actual ZCL command is made by the function below. Replace <> to “On” or “Off”.

emberAfFillCommandOnOffCluster<>()

It has to be set which Client endpoint send to which Server endpoint. emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

Send the message as unicast to the device 0x0000, the Coordinator. emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

There are two ways of implementing the functionality 1) Simple Implementation and 2) Using Events. By following both the methods we can successfully make light and switch responsive.

1) Simple Integration :   
  
#include "sl\_simple\_button.h"

#include "sl\_simple\_button\_instances.h"

#define BUTTON0 0

#define BUTTON1 1

volatile bool button0Pressed; // flag for button0

volatile bool button1Pressed; // flag for button1  
  
/\* As we are sending commands based on button change we are using this  
 Interrupt. Generally we can call the send message function within interrupt but the new version of SDK wont allow it and also not the best practice. So this function enables the flag if the button is pressed.  
  
\*/

void sl\_button\_on\_change(const sl\_button\_t \*handle)

{

if (SL\_SIMPLE\_BUTTON\_INSTANCE(BUTTON0) == handle){

if (sl\_button\_get\_state(handle) == SL\_SIMPLE\_BUTTON\_RELEASED){

button0Pressed = true;

}

}

if (SL\_SIMPLE\_BUTTON\_INSTANCE(BUTTON1) == handle){

if (sl\_button\_get\_state(handle) == SL\_SIMPLE\_BUTTON\_RELEASED){

button1Pressed= true;

}

}

}

/\*  
We Use this function for sending messages based on the flag status.  
\*/

void sendMessage()

{

EmberStatus status;

if (button0Pressed)

{

emberAfFillCommandOnOffClusterOn();

emberAfCorePrintln("Button0 is pressed");

emberAfCorePrintln("Command is zcl on-off ON");

emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

status = emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

if (status == EMBER\_SUCCESS)

{

emberAfCorePrintln("Command is successfully sent");

}

else

{

emberAfCorePrintln("Failed to send");

emberAfCorePrintln("Status code: 0x%x", status);

}

button0Pressed = false; // Resetting the Flag

}

if (button1Pressed)

{

emberAfFillCommandOnOffClusterOff();

emberAfCorePrintln("Button1 is pressed");

emberAfCorePrintln("Command is zcl on-off OFF");

emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

status = emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

if (status == EMBER\_SUCCESS)

{

emberAfCorePrintln("Command is successfully sent");

}

else

{

emberAfCorePrintln("Failed to send");

emberAfCorePrintln("Status code: 0x%x", status);

}

button1Pressed = false; // Resetting the Flag

}

}  
  
/\*  
Main Tick.

Whenever main application tick is called, this callback will be called at the end of the main tick execution

This function is called in each iteration of the main application loop and can be used to perform periodic functions. The frequency with which this function is called depends on how quickly the main loop runs. If the application blocks at any time during the main loop, this function will not be called until execution resumes.

\*/

void emberAfMainTickCallback(void)

{

if (button0Pressed || button1Pressed)

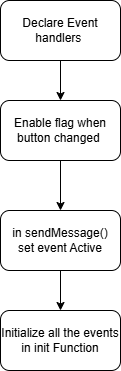
{

sendMessage();

}

}

2) Using Events :



#include "app/framework/include/af.h"

#include "sl\_simple\_button.h"

#include "sl\_simple\_button\_instances.h"

#define BUTTON0 0

#define BUTTON1 1

volatile bool button0Pressed;// Flag for Button 0

volatile bool button1Pressed;// Flag for Button 1

// autogen function

void emberAfPluginNetworkSteeringCompleteCallback(EmberStatus status,

uint8\_t totalBeacons,

uint8\_t joinAttempts,

uint8\_t finalState)

{

sl\_zigbee\_app\_debug\_println("%s network %s: 0x%02X", "Join", "complete", status);

}

// autogen function

void emberAfRadioNeedsCalibratingCallback(void)

{

sl\_mac\_calibrate\_current\_channel();

}

/\*\*\* Declare Event Handlers \*\*/

static sl\_zigbee\_event\_t sendCommandEventOn;

static sl\_zigbee\_event\_t sendCommandEventOff;

static void sendCommandEventOnHandler(sl\_zigbee\_event\_t \*event);

static void sendCommandEventOffHandler(sl\_zigbee\_event\_t \*event);

/\*\*\*\*\*/

void sl\_button\_on\_change(const sl\_button\_t \*handle)

{

if (SL\_SIMPLE\_BUTTON\_INSTANCE(BUTTON0) == handle){

if (sl\_button\_get\_state(handle) == SL\_SIMPLE\_BUTTON\_RELEASED){

button0Pressed = true;

emberAfCorePrintln("Button 0 Pressed");

}

}

if (SL\_SIMPLE\_BUTTON\_INSTANCE(BUTTON1) == handle){

if (sl\_button\_get\_state(handle) == SL\_SIMPLE\_BUTTON\_RELEASED){

button1Pressed= true;

emberAfCorePrintln("Button 1 Pressed");

}

}

}

void sendMessage(){

if(button0Pressed){

sl\_zigbee\_event\_set\_active(&sendCommandEventOn); // Setting event On active

}

if(button1Pressed){

sl\_zigbee\_event\_set\_active(&sendCommandEventOff); // Setting event OFF active

}

}

static void sendCommandEventOnHandler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status;

emberAfFillCommandOnOffClusterOn();

emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

emberAfCorePrintln("Sending Zigbee On Command...");

button0Pressed =false;

status = emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

if (status == EMBER\_SUCCESS) {

emberAfCorePrintln("Command Sent Successfully");

} else {

emberAfCorePrintln("Command Failed, Status: 0x%x", status);

}

}

static void sendCommandEventOffHandler(sl\_zigbee\_event\_t \*event)

{

EmberStatus status;

emberAfFillCommandOnOffClusterOff();

emberAfSetCommandEndpoints(emberAfPrimaryEndpoint(), 1);

emberAfCorePrintln("Sending Zigbee On Command...");

button1Pressed=false;

status = emberAfSendCommandUnicast(EMBER\_OUTGOING\_DIRECT, 0x0000);

if (status == EMBER\_SUCCESS) {

emberAfCorePrintln("Command Sent Successfully");

} else {

emberAfCorePrintln("Command Failed, Status: 0x%x", status);

}

}

void emberAfMainTickCallback(void)

{

sendMessage();

}

void emberAfMainInitCallback(void){

sl\_zigbee\_event\_init(&sendCommandEventOn, sendCommandEventOnHandler);

sl\_zigbee\_event\_init(&sendCommandEventOff,sendCommandEventOffHandler);

}

For More Information about the callbacks and functions which are used refer : <https://docs.silabs.com/zigbee/6.5/af_v2/index>  
  
  
3) Build and Flash the code into their device respectively.  
4) Create a network on Light and make sure Switch Joins the network ( Follow FormingAnd Joining Documents and use CLI commands).  
  
**Testing:**Note: Make sure the switch joins the light network.  
  
But Pressing the Button 0 on switch It turn on Led on Light and Button 1 Turn off the Led.  
