2023 tech talks WEBINAR SERIES

Welcome

Bluetooth App Development with CircuitPython

Nicola Wrachien & Scott Shawcroft

BLUETOOTH SERIES

SILICON LABS

BLUETOOTH SERIES

Presentation Will Begin Shortly

tech talks UPCOMING SESSIONS

NEW

OCT 26TH	Bluetooth App Development with CircuitPython
NOV 16TH	Enhancing Bluetooth LE Advertising Range with Novel Bits

ON DEMAND

FEB 23RD	ML in Predictive Maintenance and Safety Applications
MAR 23RD	Unboxing: What's New With Bluetooth
APR 20TH	What's New with Bluetooth Mesh 1.1
MAY 18TH	Bluetooth Portfolio: What's Right for Your Application
JUN 15TH	The Latest in HADM With Bluetooth LE

We will begin in: 0:00

Agenda

About CircuitPython

Architecture

Supported Boards

Port Features

Example: Bluetooth LE Application

Sample Code

Conclusion

About CircuitPython

Origin of CircuitPython

PYTHON IS VERY HIGH LEVEL

- Good for beginners
- Lots of libraries
- Batteries included → fast development
- Not good for MCUs
  - Limited access to hardware resource
  - Memory hungry

MICROPYTHON WAS DEVELOPED

- Optimized for 32-bit MCUs
- Some less-used features not implemented
- REPL (read-eval-print loop) console over UART
- Target Python 3.4 language features
- Limited standard library support

CIRCUITPYTHON, SIMPLIFIED

- Fork of MicroPython
- Focused on students, beginners, ease of use
- Unified hardware-access APIs
- Initially: on different, less powerful, MCUs than MicroPython (now many powerful MCUs supported too)

CircuitPython: Advantages and Drawbacks

ADVANTAGES

- **Very high-level language**
  - Few lines to create relatively complex programs
- **Beginner Friendly:**
  - Easy to use, with smooth learning curve
  - Automatic memory management, with garbage collector
  - No pointers
  - Tons of libraries
  - Tons of examples and guides
- **Cross-platform compatible**
  - The same project can run in different MCUs
  - Many projects can run on Linux SBCs via Blinka
- **No compilation time**
- **Many more Python devs than C devs**
- **Simple setup**

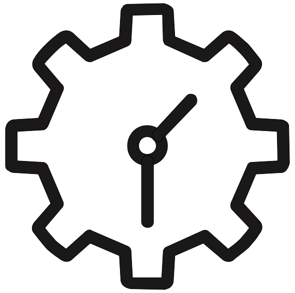
DRAWBACKS

- **Interpreted language, slow, memory intensive**
  - Large flash footprint by default
- **Less control on hardware**
  - Hard to use MCU-specific peripherals without Python driver
- **Some hidden issues, which might be frustrating sometimes**
  - How does “0xfor x in (1, 2, 3)” eval?
  - Errors found during runtime rather than compile time (e.g. modules not found)
  - Hidden bugs that might have been caught at compile time if strong typed

SILICON LABS

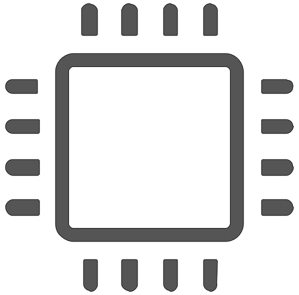
C-code Performance Not Always a Must

WHY CIRCUITPYTHON?



QUICK POC CREATION

Focus on the application behavior and Time to Market



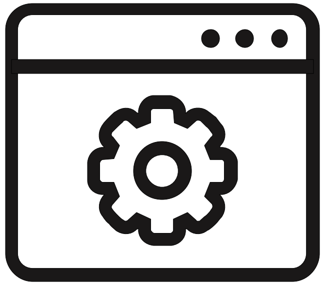
STRONG MCU

~80 MHz Cortex M33 is fast enough to handle higher level languages than C at a good speed

REAL-WORLD APPLICATIONS

E.g., On/Off relay or a thermostat

Response time in ~ms is more than enough



USING EXISTING LIBRARIES

The CPU-intensive work can be done by existing libraries, written in C.

The impact of Python implementation is reduced.

SILICON LABS

About Silicon Labs

SILICON LABS FOCUS: IOT MARKET

MANY PRODUCT VARIANTS

Different formfactors
Cost
Performance (CPU, RAM, FLASH, MVP)
Peripheral set
Etc.

DIVERSE PROTOCOLS

Bluetooth LE
Zigbee
Matter
Z-Wave
Etc.

WIDE MARKET

Smart Home
Industrial IoT
Smart Cities
Smart Retails
Connected Health

Addressing Makers Ecosystem

Addressing the maker community's need to interface with a broad audience **from beginners…up to experts.**

CURRENT SILICON LABS DEVELOPMENT ENVIRONMENT

- **Our main SDK is the Gecko SDK (GSDK)**
  - Largely based on C
  - Hundreds of different files
  - Maintained by Silicon Labs
  - No or limited third party contribution
- **Our Main Development Tool is Simplicity Studio**
  - C or C++ projects
- **Bluetooth LE is a rather complex protocol**

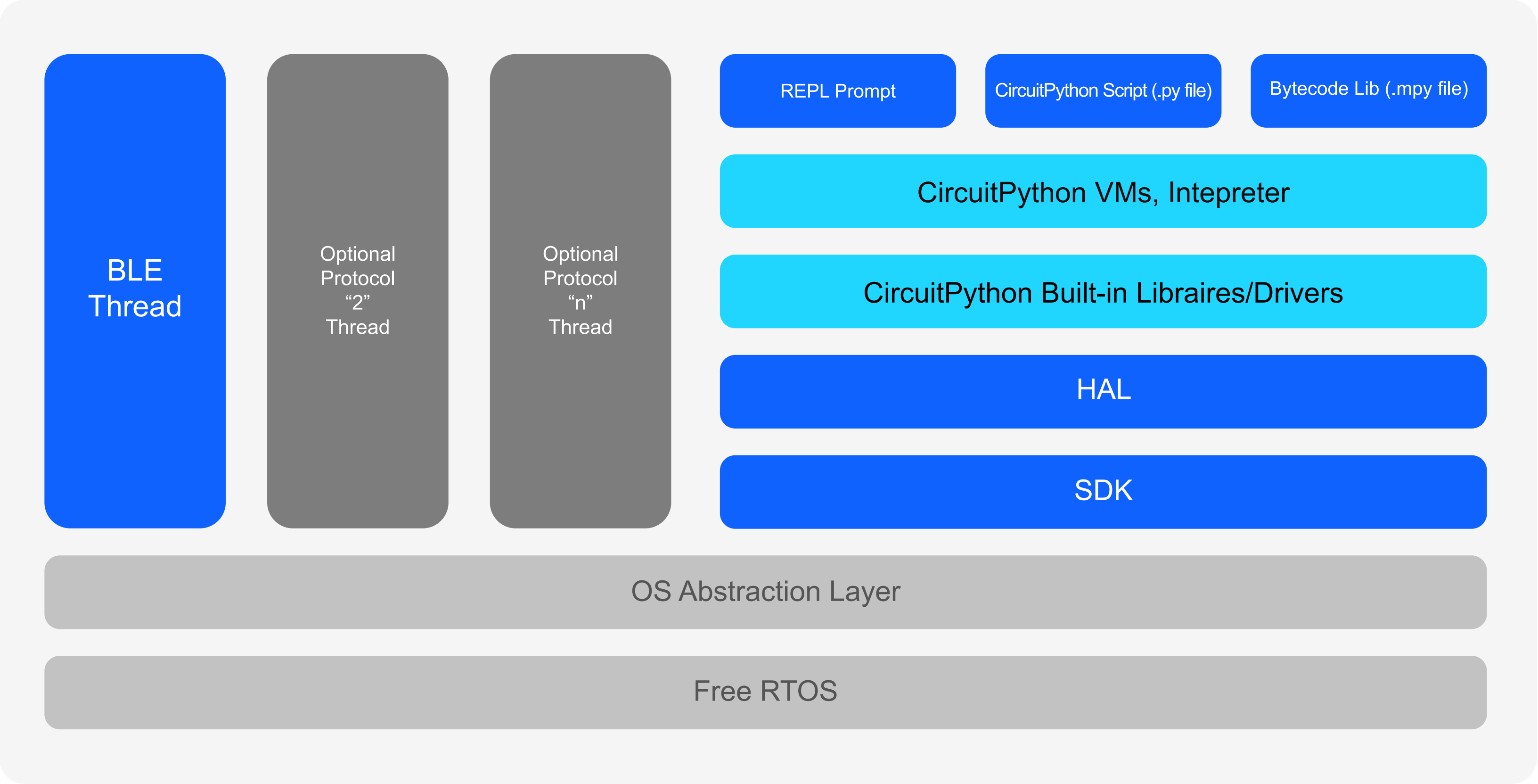
WELL KNOWN ECOSYSTEMS, NEEDS TO BE SUPPORTED

circuit python

Architecture

Architecture

- Multithread architecture using Free RTOS (other RTOS can be used thanks to OS abstraction layer)
- Less issues in handling time critical protocol-related tasks.
- Easier to add other protocols (Dynamic multi protocol can be supported as well)



SILICON LABS

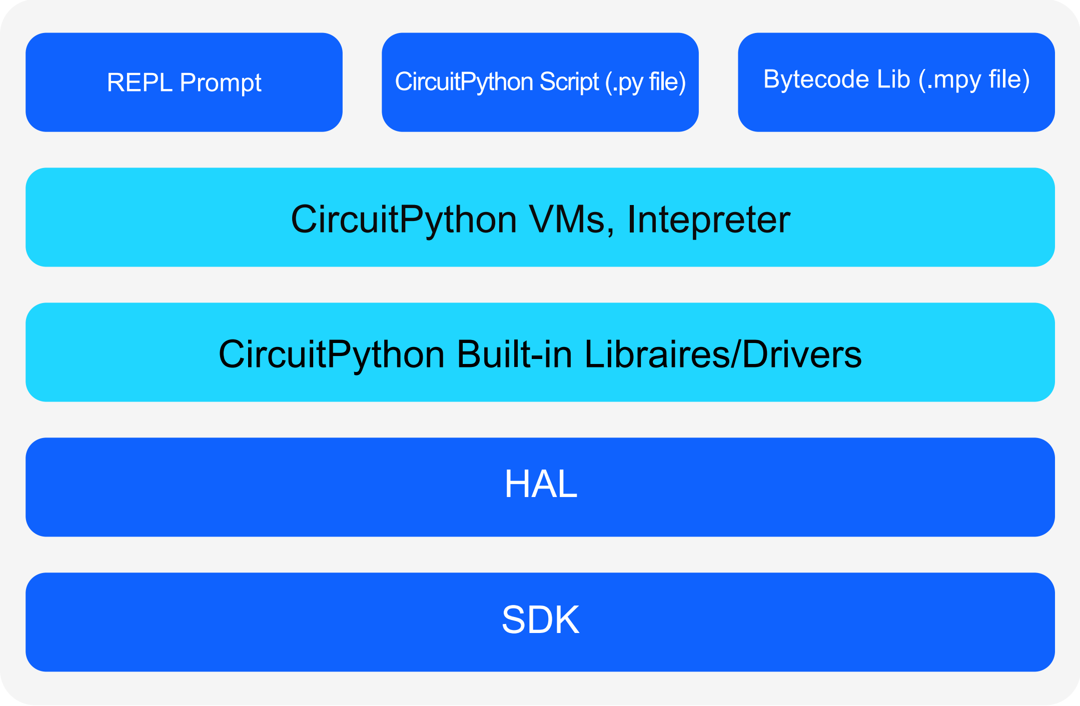
Architecture

Application layer

- **Python code and libraries for the project.**
  Code and libraries loaded from the filesystem or REPL (Read-Eval-Print Loop) console.

- **REPL console**
  Command-line interface that allows developers to interactively test and debug their code on the microcontroller.

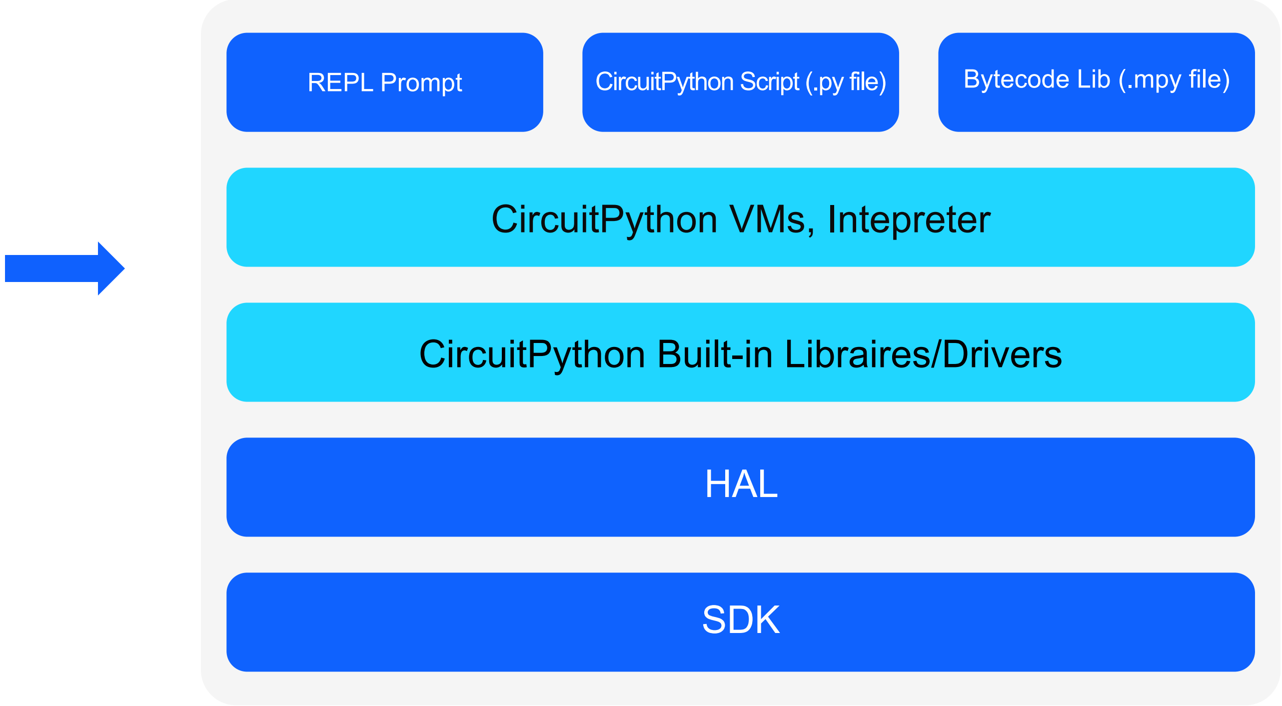
  REPL runs only after the code.py finishes.



Architecture

CircuitPython core layer

- **Core libraries:**
  Standard Python libraries including modules for math, string manipulation, file I/O, and more…
- **Runtime environment:**
  Provides the necessary infrastructure to execute Python code on the microcontroller, including memory management, garbage collection, and exception handling.
- **Python Interpreter:**
  Interprets py code, compiles and feeds it to the VM.
- **Python Virtual Machine:**
  Executes python bytecode.



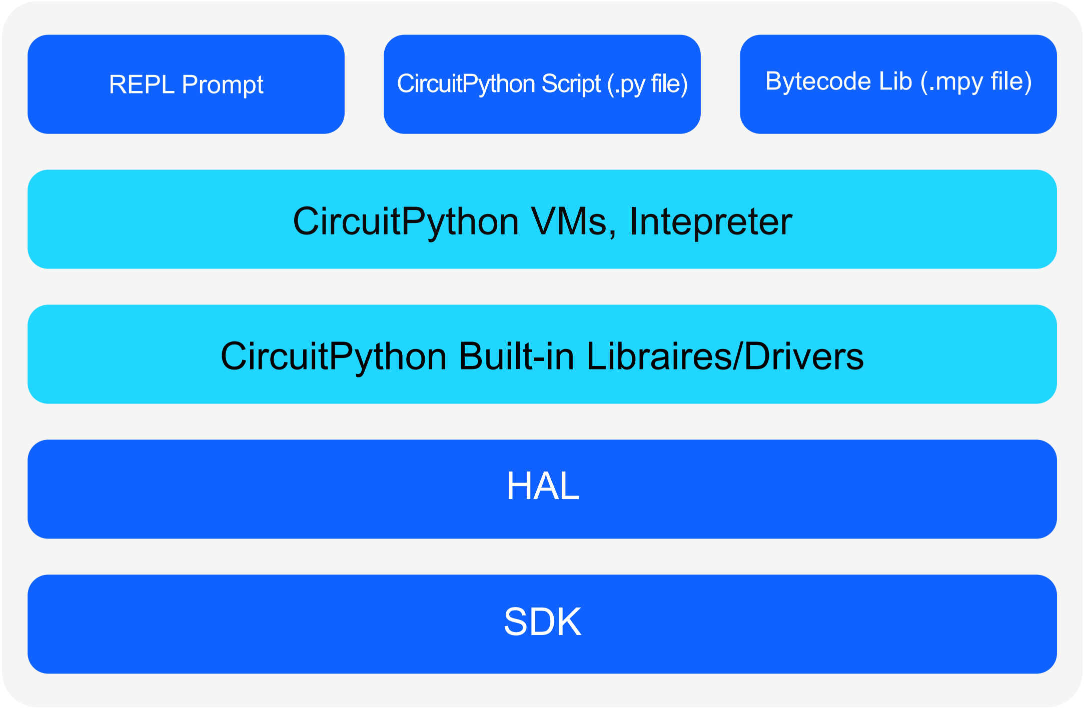
Architecture

Hardware abstraction layer (HAL)

- Provides a consistent interface to interact with the MCU hardware.
- Set of HAL API provided by CircuitPython.
- MCU-specific API implementation

SDK

- Provides stacks, and low-level MCU-specific drivers



SILICON LABS

Supported Boards

Supported Boards

SPARKFUN THINGPLUS MATTER

- **xG24 based, supporting BLE**
- **On-board debugger**
- **3rd party hardware support**
  - Qwiic connector
- **Lowest price point**
- **Feather Formfactor**
- **SD Card slot**

XG24 EXPLORER KIT

- **xG24 based, supporting BLE**
- **On-board debugger**
- **3rd Party Hardware Support**
  - Qwiic connector
- **MikroBus Connector**

XG24 DEVELOPMENT KIT

- **xG24 based, supporting BLE**
- **On-board debugger**
- **3rd Party Hardware Support**
  - Qwiic connector
- **On-Board sensors**
- **Impressive out-of-the-box demos**
- **External Flash**

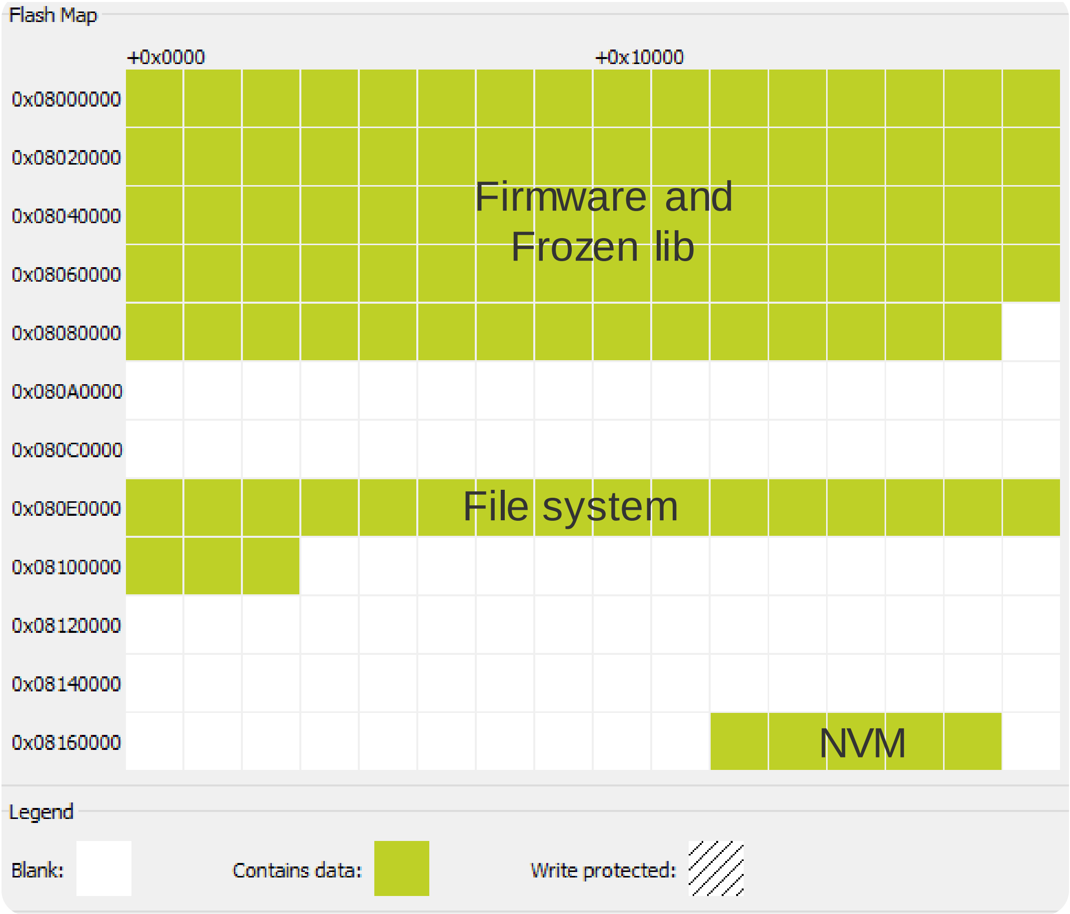
SILICON LABS

Port Features

Flash Memory and FileSystem

- CircuitPython uses a file system to store code/data:
  - Internal Flash
  - External Flash
  - External SD card (adafruit_sdcard)

- Modules and code loaded to RAM
  - ➔ high RAM usage
  - Frozen lib are integrated in the firmware
  - ➔ no need to load them in RAM

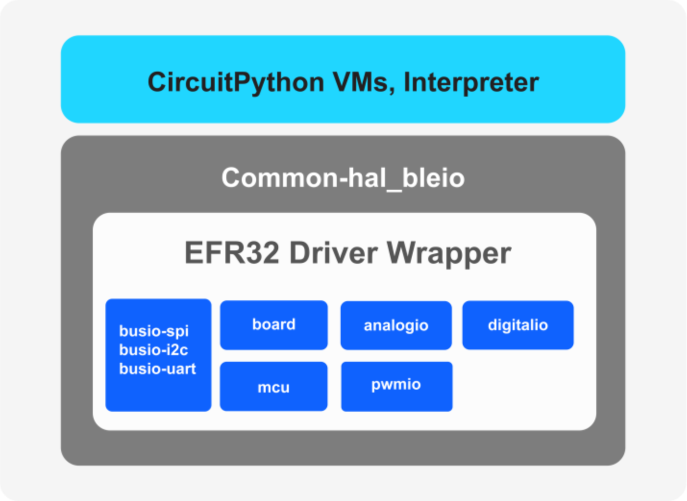


SILICON LABS

Internal Peripheral Support

Low level drivers made in C supporting internal MCU peripherals:

- Serial interfaces: I2C, SPI, UART (**busio**)
- Analog functions (ADC, DAC) (**analogio**)
- GPIOs (**digitalio**)
- PWM (**pwmio**)
- RTC (**rtc**)
- NVM (**microcontroller.nvm**)



CircuitPython allows flexible configuration of pins for peripheral communication.

- This uses Silicon Labs EFR32 devices' ability of mapping any peripheral to almost any GPIO.

Board's default UART pin assignment:

```
uart = busio.UART(board.TX, board.RX, baudrate=9600)
```

Routing to a different pin:

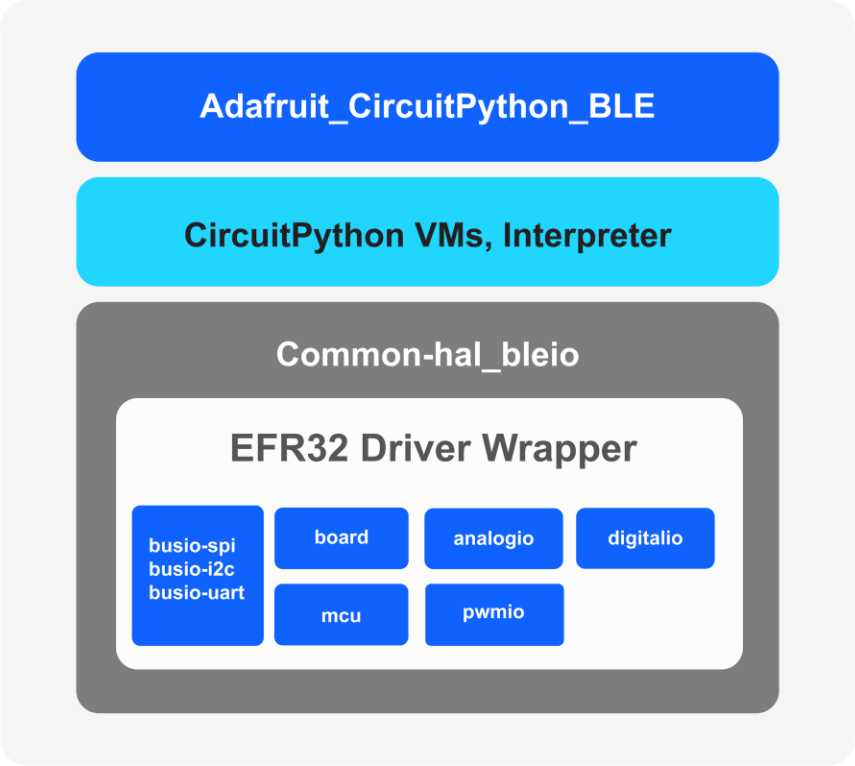
```
uart = busio.UART(board.PB1, board.PB2, baudrate=9600)
```


SILICON LABS

External Peripheral Support

Adafruit CircuitPython Library Bundle

- Collection of python libraries/examples for over 300 devices (display, sensors, etc).
- Allow to use any external peripheral on any MCU, having some internal peripheral sets.
- Based on lower level drivers coded in C for internal peripheral support.




```
import board                                              # import board definitions and peripherals
from adafruit_bme280 import basic as adafruit_bme280     # import Adafruit Library Bundle BME280 driver

i2c = board.I2C()                                         # create i2c object required by the sensor driver
bme280 = adafruit_bme280.Adafruit_BME280_I2C(i2c)        # create bme280 driver object

print("\nTemperature: %0.1f C" % bme280.temperature)     # read and print temperature
```

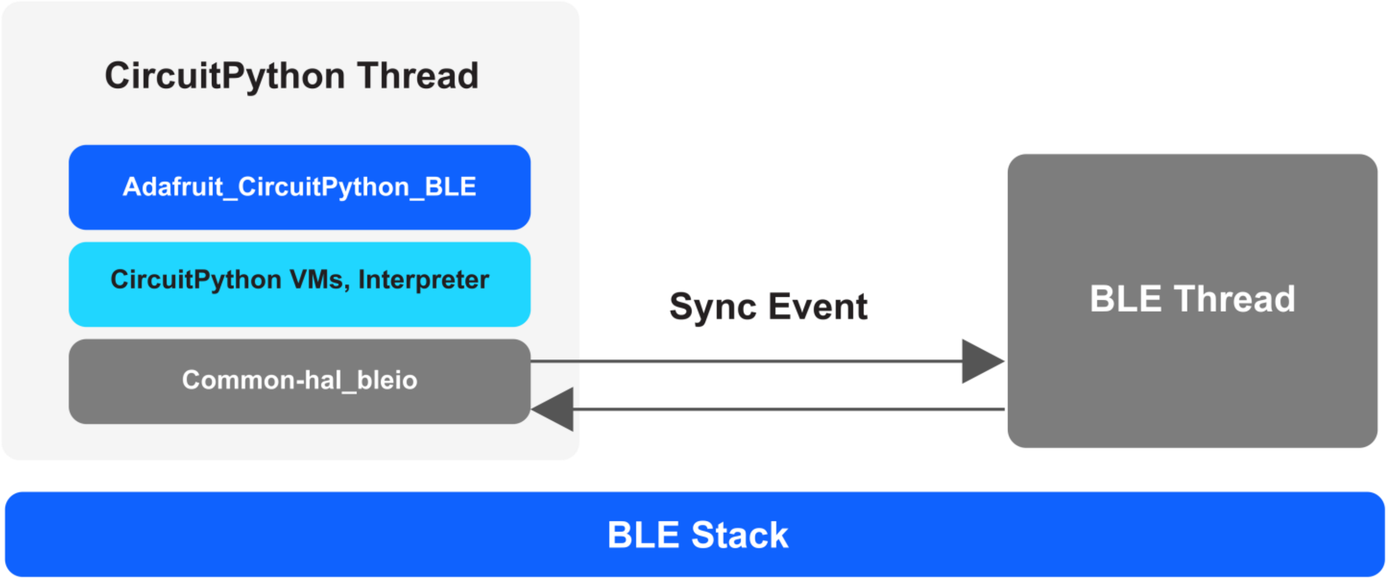
Bluetooth LE Support

Adafruit_Circuitpython_BLE

- python module providing high level easy to use APIs for BLE.
- Can be both external or frozen lib.
- built over _bleio (in C)

_bleio

- _bleio module:
  provides necessary low-level functionality for BLE,
  interact with events from BLE thread.
- dynamic GATT table support



Start Advertising with six lines of code!

```
from adafruit_ble import BLERadio
from adafruit_ble.advertising import Advertisement
ble = BLERadio()                                            # init BLE
adv = Advertisement()                                       # create advertisement object
adv.complete_name="Silabs CircuitPython"                    # set complete name
ble.start_advertising(advertisement = adv, interval = 1)    # start advertising
```

Performance Comparison

Performance Comparison

- **Hard to do a fair comparison**
  - Which optimization level was used?
  - Which operation?
  - Which libraries?

xG24 DEV kit

V.S.

xG24 DEV kit

SILICON LABS

Performance Comparison

- **Inefficient bubble-sort algorithm + optimized library sort function of a descendent array**

Circuitpython code

```python
# Bubble Sort speed test
import time
import microcontroller
def bubbleSort(arr):
    n = len(arr)
    for i in range(n-1):
        for j in range(0, n-i-1):
            # Swap if out of order
            if arr[j] > arr[j + 1]:
                swapped = True
                arr[j], arr[j + 1] = arr[j + 1], arr[j]
        if not swapped:
            # exit if no swap
            return
# Test speed
print("Bubble sort example")
print("USING CUSTOM FUNCTION:")
print("CPU Speed:", microcontroller.cpu.frequency / 1000000.0,"MHz")
numTests = 16
arraySize = 64
startTime= time.monotonic_ns()
for i in range(0, numTests):
    arr = list(range(arraySize, 0, -1))
    bubbleSort(arr)
stopTime = time.monotonic_ns()
avgMs = (stopTime - startTime) * 1e-6 / numTests
print("Avg time sorting ", arraySize,"elements is", avgMs, "ms");
print("CPU cycles ", avgMs * microcontroller.cpu.frequency / 1000);
#built-in function
startTime= time.monotonic_ns()
for i in range(0, numTests):
    arr = list(range(arraySize, 0, -1))
    arr.sort()
stopTime = time.monotonic_ns()
avgMs = (stopTime - startTime) * 1e-6 / numTests
print("USING LIBRARY FUNCTION:")
print("Avg time sorting ", arraySize,"elements is", avgMs, "ms");
print("CPU cycles ", avgMs * microcontroller.cpu.frequency / 1000);
```

C code

(Simplicity Studio empty C project + test)

```c
#include "em_cmu.h"
#include <stdbool.h>
#include
#include
#include
#define
int arra
void bub
{
    bool
    for



    }
}
int intC
{
    retur
}
void tes
{
  unsign
  printf
  SysTic
  SysTic
  SysTick->CTRL = 5;
  printf("CPU Speed: %d MHz\r\n",   cpuFreq / 1000000);
```

```c
  size_t numTests = 16;
  unsigned int timeStart, timeStop;
  unsigned int avgTimeUs;
  timeStart = SysTick->VAL;
  for (size_t test = 0; test < numTests; test++) {
    // init array
    for (size_t i = 0; i < ARRAY_SIZE; i++) {
      array[i] = ARRAY_SIZE - i;
    }
    // sort it
    bubbleSort(array, ARRAY_SIZE);
  }
  timeStop = SysTick->VAL;
  avgTimeUs = (timeStart - timeStop) * 1000000.0f / cpuFreq / numTests;
  //
  pr
  pr
  pr
  //
  ti
  fo




    // sort it
    qsort(array, ARRAY_SIZE, sizeof(array[0]), intCmp);
  }
  timeStop = SysTick->VAL;
  avgTimeUs = (timeStart - timeStop) * 1000000.0f / cpuFreq / numTests;
  //
  printf("USING LIBRARY FUNCTION:\r\n");
  printf("Avg time sorting %u elements is %u us\r\n", ARRAY_SIZE, avgTimeUs);
  printf("CPU cycles %u\r\n", (timeStart - timeStop) / numTests);
```

…and some tens more files…

(autogenerated, still require user configuration – UART, clock, GPIOs)

Performance Results

```
Bubble sort example
CPU Speed: 78 MHz
USING CUSTOM FUNCTION:
Avg time sorting 64 elements is 296 us
CPU cycles 23161
USING LIBRARY FUNCTION:
Avg time sorting 64 elements is 80 us
CPU cycles 6290
```

```
Bubble sort example
USING CUSTOM FUNCTION:
CPU Speed: 78.0 MHz
Avg time sorting  64 elements is 70.4345 ms
CPU cycles  5.49389e+06
USING LIBRARY FUNCTION:
Avg time sorting  64 elements is 2.07519 ms
CPU cycles  161865.0
```

C vs **CircuitPython** bubble sort implementation: 296 microseconds vs 70 ms: **236 times slower**

“Only” 25 times slower if using libraries. Libraries are 35 times fasters.

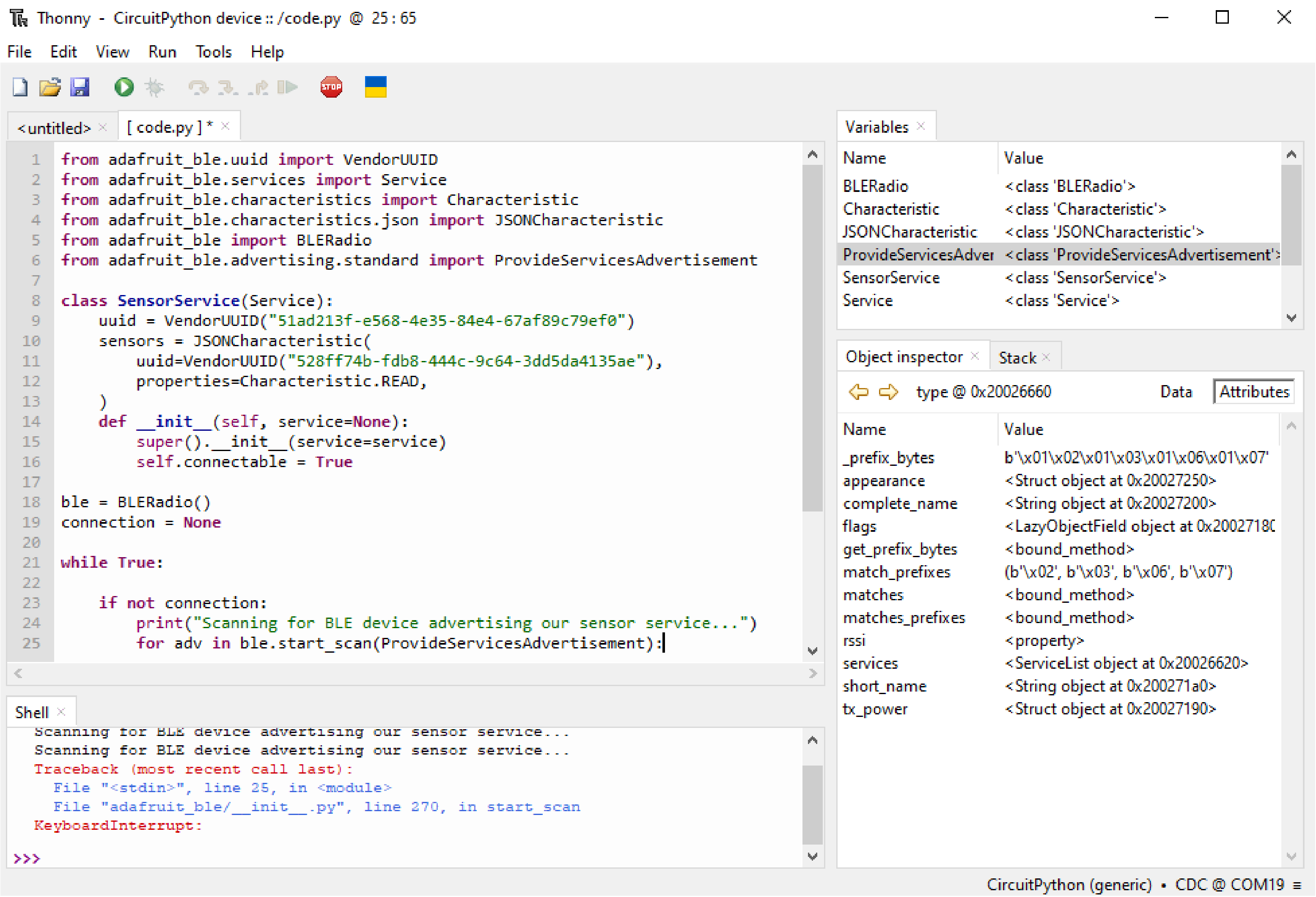
➔ Don’t reinvent the wheel. Use libraries whenever possible!

SILICON LABS

Bluetooth LE Example

Suggested Development Environment

- **We suggest Thonny**
  - Syntax highlight
  - REPL console output
  - Directly uploads, even without native USB
  - Additional features:
    - Variable list
    - Program tree
    - Object inspector



BLE Example features

- **Measure temperature and humidity (xG24 Development Board)**
- **Collector automatically finds the sensor board, connects, and prints the results. (Sparkfun ThingPlus Matter board).**

- Advertises with a specific payload
- Measures humidity and temperature
- Accepts connections from a central device

- Scans for devices with the Sensor Service
- Connects to the device
- Reads measurements

Simple Thermometer + Humidity Sensor Application

Bluetooth LE peripheral device

```python
from adafruit_ble.uuid import VendorUUID
from adafruit_ble.services import Service
from adafruit_ble.characteristics import Characteristic
from adafruit_ble.characteristics.json import JSONCharacteristic
import time
import sensor
import board
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement

class SensorService(Service):
    uuid = VendorUUID("51ad213f-e568-4e35-84e4-67af89c79ef0")
    sensors = JSONCharacteristic(
        uuid=VendorUUID("528ff74b-fdb8-444c-9c64-3dd5da4135ae"),
        properties=Characteristic.READ | Characteristic.NOTIFY,
    )
    def __init__(self, service=None):
        super().__init__(service=service)
        self.connectable = True

#init sensors
i2c = board.I2C()
sensor.init(i2c)
#init BLE
ble = BLERadio()
service = SensorService()
advertisement = ProvideServicesAdvertisement(service)
advertisement.short_name="SilabsCP"
def measure():
    temperature = sensor.temperature()
    humidity = sensor.humidity()
    return {"temperature": temperature,"humidity":humidity}

while True:
    print("Advertise services")
    ble.stop_advertising()
    ble.start_advertising(advertisement)
    print("Waiting for connection...")
    while not ble.connected:
        pass
    print("Connected")
    while ble.connected:
        service.sensors = measure()
        time.sleep(0.25)
    print("Disconnected")
```

Bluetooth LE central device

```python
from adafruit_ble.uuid import VendorUUID
from adafruit_ble.services import Service
from adafruit_ble.characteristics import Characteristic
from adafruit_ble.characteristics.json import JSONCharacteristic
from adafruit_ble import BLERadio
from adafruit_ble.advertising.standard import ProvideServicesAdvertisement

class SensorService(Service):
    uuid = VendorUUID("51ad213f-e568-4e35-84e4-67af89c79ef0")
    sensors = JSONCharacteristic(
        uuid=VendorUUID("528ff74b-fdb8-444c-9c64-3dd5da4135ae"),
        properties=Characteristic.READ | Characteristic.NOTIFY,
    )
    def __init__(self, service=None):
        super().__init__(service=service)
        self.connectable = True

ble = BLERadio()
connection = None

while True:

    if not connection:
        print("Scanning for BLE device advertising our sensor service...")
        for adv in ble.start_scan(ProvideServicesAdvertisement):
            print(adv.services)
            if SensorService in adv.services:
                connection = ble.connect(adv)
                print("Connected")
                break
        ble.stop_scan()

    if connection and connection.connected:
        service = connection[SensorService]
        while connection.connected:
            print("Sensors: ", service.sensors)
```

```
Scanning for BLE device advertising our sensor service...
<BoundServiceList: UUID('51ad213f-e568-4e35-84e4-67af89c79ef0')>
Connected
Sensors:  {'humidity': 50.625, 'temperature': 27.839}
Sensors:  {'humidity': 50.625, 'temperature': 27.839}
Sensors:  {'humidity': 50.625, 'temperature': 27.861}
Sensors:  {'humidity': 50.625, 'temperature': 27.861}
Sensors:  {'humidity': 50.625, 'temperature': 27.861}
```

- Very few lines of code for both applications!

SILICON LABS

Sample Code

Sample Code

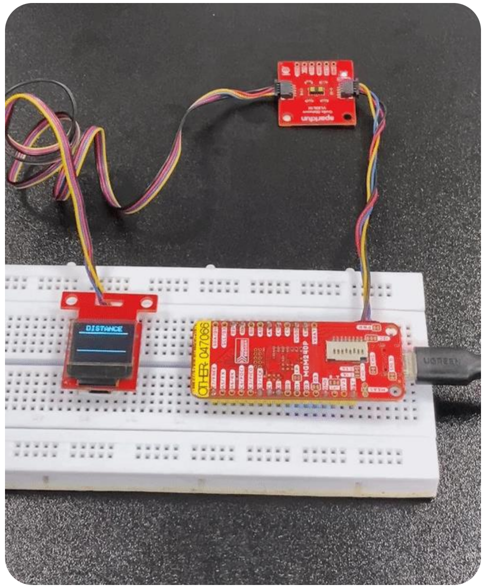
- In https://github.com/SiliconLabs/circuitpython_applications/tree/main we released some examples

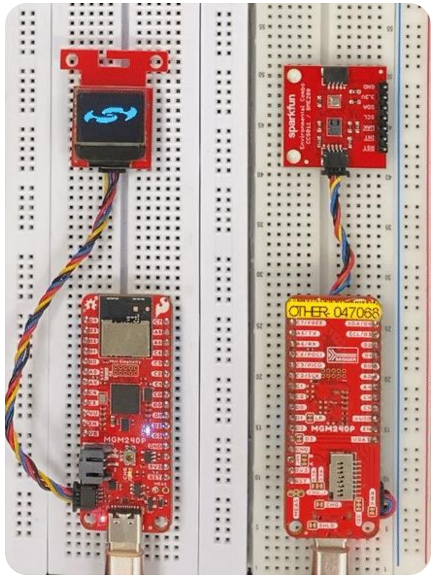
No	Example name	Link to example
1	CircuitPython - Bluetooth - Distance Monitor (VL53L1X)	Click Here
2	CircuitPython - Bluetooth - Environmental Sensing (CCS811/BME280)	Click Here
3	CircuitPython - Bluetooth - Neopixel Humidity Gauge (SHTC3)	Click Here
4	CircuitPython - Bluetooth - Light Detector (AS7265x)	Click Here
5	CircuitPython - Non-Wireless Display Demo (IS31FL3741)	Click Here
6	CircuitPython - RGB Display Drawing (ILI9341)	Click Here
7	CircuitPython - Temperature and Humidity Monitor with LED Matrix Display (SI2071/IS31FL3741)	Click Here
8	CircuitPython - xG24 Dev Kit Sensors (ILI9341)	Click Here

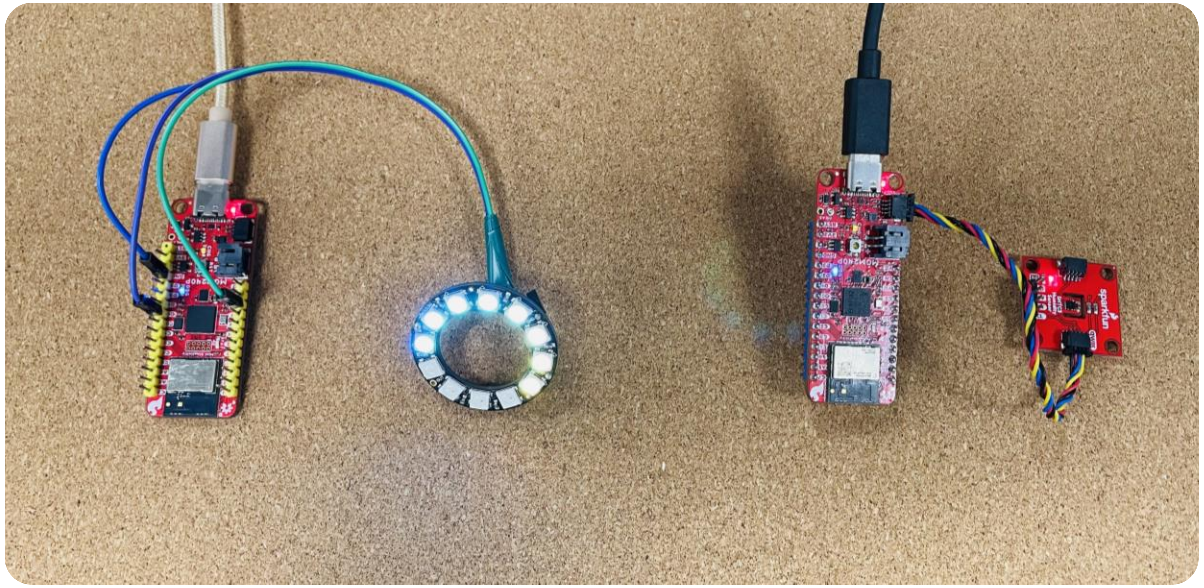
Bluetooth LE-based

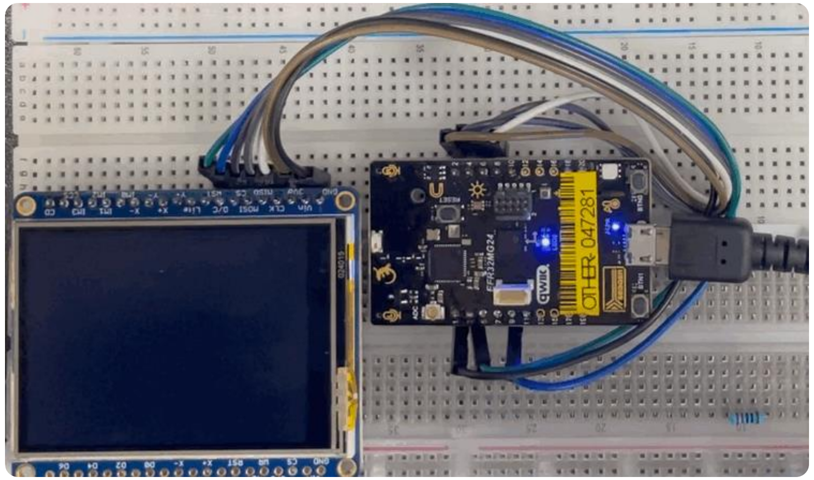
SILICON LABS

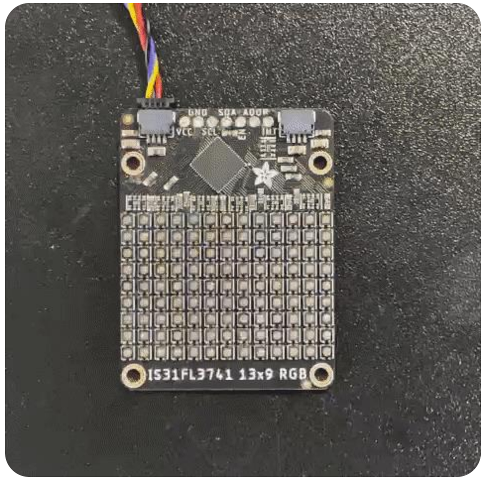
Some examples











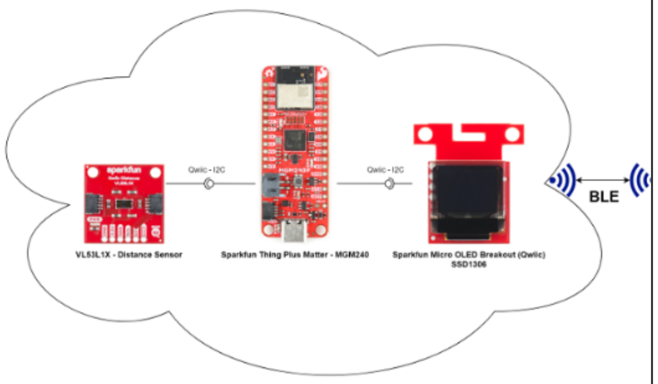
Sample Code

CircuitPython - Bluetooth - Distance Monitor (V

Overview

This project shows a demonstration of a Bluetooth Low Energy distance monitor system using Spa
development kit and the integrated CircuitPython BLE Stack.

The block diagram of this application is shown in the image below:



Hardware Required

- SparkFun Thing Plus Matter - MGM240P
- SparkFun Distance Sensor Breakout - 4 Meter, VL53L1X (Qwiic)
- OLED Display - SSD1306

Connections Required

The sensor and OLED display can easily connected with Sparkfun Thing Plus for Matter - MGM240

- Runtime operation

- GATT database
  - [Service] Distance Monitor
    - [Char] Lower Threshold Value - threshold_value_lower
      - [R] Get lower threshold value (mm)
      - [W] Set lower threshold value (mm)
    - [Char] Upper Threshold Value - threshold_value_upper
      - [R] Get upper threshold value (mm)
      - [W] Set upper threshold value (mm)
    - [Char] Threshold Mode - threshold_mode
      - [R] Get threshold mode (0-2)
      - [W] Set threshold mode (0-2)
    - [Char] Range Mode - range_mode
      - [R] Get configured range mode (0-1)
      - [W] Set range mode (0-1)
    - [Char] Notification Status - notification_status
      - [R] Get configured notification status (0-2)
      - [W] Set notification status (0-2)

u need to install **Thonny** editor and then follow the steps below:

nding CircuitPython binary for your board. You can visit circuitpython.org/downloads to download the binary.

s in this repository require CircuitPython v8.2.0 or higher.

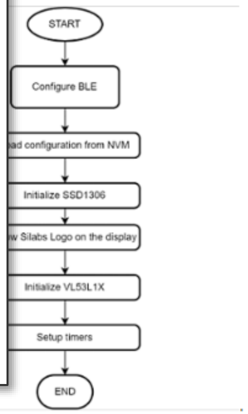
ry libraries from Adafruit CircuitPython bundle. You can download the bundle from here. The libraries that used in this
ersion are list in this table below.

	Version
uf	1.6.1
6	2.12.2
	1.1.10

aries of the lib folder to the CircuitPython device. The binary files should not be uploaded to lib folder in the device, they
me hierarchy as the code.py file.

of the code.py and paste it to the code.py file on the CircuitPython device.

the board.



SILICON LABS

Conclusions

Conclusions

- **Silicon Labs is now actively supporting Circuit Python on xG24 boards**
  - Main features implemented:
    - Digital GPIO support
    - Analog functions (DAC, ADC)
    - Serial interfaces (UART, SPI, I2C)
    - NVM and filesystem (including SD support)
    - Bluetooth LE

- **The architecture allows for easy feature extension**
  - New board and SoC support
  - Adding support for additional protocols.

- **CircuitPython allows writing complex programs in few lines**

➔ Good for learning and for quick PoC designs

SILICON LABS

2023 tech talks WEBINAR SERIES

Thank You

BLUETOOTH SERIES

Watch ON DEMAND

Q&A

BLUETOOTH SERIES

BLUETOOTH SERIES

tech talks UPCOMING SESSIONS

NEW

OCT 26TH	Bluetooth App Development with CircuitPython
NOV 16TH	Enhancing Bluetooth LE Advertising Range with Novel Bits

ON DEMAND

FEB 23RD	ML in Predictive Maintenance and Safety Applications
MAR 23RD	Unboxing: What's New With Bluetooth
APR 20TH	What's New with Bluetooth Mesh 1.1
MAY 18TH	Bluetooth Portfolio: What's Right for Your Application
JUN 15TH	The Latest in HADM With Bluetooth LE