@1stship 2019年12月23日に更新

# Wio LTE消費電力の研究

SORACOM WioLTE

## はじめに

良い感じの電力計を一時的にゲットしたので、
Grove IoT Starter kit for SORACOMに入っているセンサー類やLTEモジュールを動かした時にどのくらいの電力を使っているかをまとめました。

結果だけを見たい方はこちら

## 使用機器

測定対象

- Grove IoT Starter kit for SORACOM
- Wio Extension - RTC

測定器

- PW3335-04 パワーメータ

## 測定にあたって

測定にあたって困ったことが分かりまして、どうも書き込んだプログラムによってベースとなる消費電力が変わるようです。

あるプログラムだと初期化後の消費電力が270mWなのに、別のプログラムだと同じ処理なのに340mWになったりします。原因は分かりませんが、プログラムサイズや使っているメモリなどが電力に影響しているのだと思われます。

そのため、電力の測定はそのプログラムにおける動作させていない状態も測定し、動作状態と非動作状態を比較することとします。結局は実際の用途のプログラム書いた状態で消費電力測定する必要があるのですが、このくらいになるという目安にはなるでしょう。

## Grove システム対応モジュール

GroveシステムとはSeeed社が提供しているインタフェース規格です。専用のコネクタで対応モジュールを接続することにより、様々なセンサーやブザーなどを使うことができます。

対応モジュールは非常に多く、スイッチサイエンス社のページで確認すると2019/12/21現在で、
センサーは74個、
音・表示・動作は31個あります。用途にあったデバイスを探して、手軽に試してみることができます。

今回はその中でGrove IoT Starter kit for SORACOMに同梱されている 7 つのモジュールを対象に測定します。

## -1. 温湿度センサー

温度と湿度を測定できるセンサーです。

2秒ごとに温度センサーを取得する状態で 1 分間 3 回測定しました。温度センサーを動作させていない状態、温度センサーを接続していない状態と比較しています。

```
#include <WioLTEforArduino.h>

#define SENSOR_PIN    (WIOLTE_D38)
#define INTERVAL    (2000)

unsigned long uptime;
unsigned int count = 0;

void setup()
{
  TemperatureAndHumidityBegin(SENSOR_PIN);
}

void loop()
{
  count++;
  if (count <= 45){
    float temp;
    float humi;
    TemperatureAndHumidityRead(&temp, &humi);
    if (count == 45){
      SerialUSB.print("Current humidity = ");
      SerialUSB.print(humi);
      SerialUSB.print("%  ");
      SerialUSB.print("temperature = ");
      SerialUSB.print(temp);
      SerialUSB.println("C");
      SerialUSB.println("Finish Measure");
    }
  } else if (count <= 90){
    if (count == 90){
      SerialUSB.println("Finish no operation state");
    }
  } else {
    if (count == 135){
      count = 0;
      SerialUSB.println("Finish no power state");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}

// 省略：TemperatureAndHumidityBegin
// 省略：TemperatureAndHumidityRead
```

| 項目 | 温湿度取得(30回) | 取得なし | 未接続 |
| --- | --- | --- | --- |
| 項目 | 温湿度取得(30回) | 取得なし | 未接続 |
| 1回目 | 5.2602mWh | 4.8506mWh | 4.8279mWh |
| 2回目 | 5.2229mWh | 4.8079mWh | 4.8305mWh |
| 3回目 | 5.2232mWh | 4.8562mWh | 4.8346mWh |
| 平均 | 5.2354mWh | 4.8382mWh | 4.8310mWh |

センサーを接続しているだけではほぼ影響ありませんが、温湿度を取得すると0.3972mWh消費電力量が増加しています。温湿度測定１回あたり13.24μWhの電力量を消費することになります。

## -2. 加速度センサー

3軸で加速度を測定できるセンサーです。

100ミリ秒ごとに１分間加速度センサーの値を取得します。比較のため、動作していない状態、I2Cに電源を入れていない状態も測定します。

```
#include <WioLTEforArduino.h>
#include <ADXL345.h>          // https://github.com/Seeed-Studio/Accelerometer_ADXL345

#define INTERVAL    (100)

WioLTE Wio;
ADXL345 Accel;
unsigned long uptime;
unsigned int count = 0;

void setup()
{
  delay(200);
  Wio.Init();
  Wio.PowerSupplyGrove(true);
  delay(500);
  Accel.powerOn();

  SerialUSB.println("### Setup completed.");
}

void loop()
{
  int x;
  int y;
  int z;

  count++;
  if (count <= 900){
    Accel.readXYZ(&x, &y, &z);
    if (count == 900){
      SerialUSB.print(x);
      SerialUSB.print(' ');
      SerialUSB.print(y);
      SerialUSB.print(' ');
      SerialUSB.println(z);
      SerialUSB.println("Finish Measure");
    }
  } else if (count <= 1800){
    if (count == 1800){
      Wio.PowerSupplyGrove(false);
      SerialUSB.println("Finish no operation state");
    }
  } else {
    if (count == 2700){
```

```
      count = 0;
      Wio.PowerSupplyGrove(true);
      delay(500);
      Accel.powerOn();
      SerialUSB.println("Finish no power state");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}
```

| 項目 | 加速度取得(600回) | 取得なし | 電源OFF |
|---|---|---|---|
| 1回目 | 5.3696mWh | 5.3747mWh | 5.3605mWh |
| 2回目 | 5.3749mWh | 5.3646mWh | 5.3362mWh |
| 3回目 | 5.3559mWh | 5.3674mWh | 5.3362mWh |
| 平均 | 5.3668mWh | 5.3689mWh | 5.3443mWh |

センサーの接続や値の取得によって消費電力にはほぼ影響ありません。
電源をOFFにするとわずかに下がる程度です。(消費電力1.476mW減)

## -3. 超音波距離センサー

距離を測定できるセンサーです。

100ミリ秒ごとに超音波距離センサーから距離を取得します。遠くを測定している時(537cm)と、近くを測定している時(0cm近く)のそれぞれについて測定し、測定していない時、センサー未接続の時と比較します。

```
#include <WioLTEforArduino.h>
#include <Ultrasonic.h>      // https://github.com/Seeed-Studio/Grove_Ultrasonic_Ranger

#define ULTRASONIC_PIN  (WIOLTE_D38)
#define INTERVAL    (100)

Ultrasonic UltrasonicRanger(ULTRASONIC_PIN);

unsigned long uptime;
unsigned int count = 0;

void setup()
{
}

void loop()
{
  count++;
  if (count <= 900){
    long distance;
    distance = UltrasonicRanger.MeasureInCentimeters();
    if (count == 900){
      SerialUSB.print(distance);
      SerialUSB.println("[cm]");
      SerialUSB.println("Finish Measure long range");
    }
  } else if (count <= 1800){
    long distance;
    distance = UltrasonicRanger.MeasureInCentimeters();
    if (count == 1800){
      SerialUSB.print(distance);
      SerialUSB.println("[cm]");
```

```
      SerialUSB.println("Finish Measure short range");
    }
  } else if (count <= 2700){
    if (count == 2700){
      SerialUSB.println("Finish no operation state");
    }
  } else {
    if (count == 3600){
      count = 0;
      SerialUSB.println("Finish no power state");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}
```

| 項目 | 遠くを測定(600回) | 近くを測定(600回) | 測定せず | 未接続 |
|---|---|---|---|---|
| 1回目 | 5.6364mWh | 6.2156mWh | 5.4347mWh | 5.2328mWh |
| 2回目 | 5.6388mWh | 6.2161mWh | 5.4732mWh | 5.2400mWh |
| 3回目 | 5.6334mWh | 6.2214mWh | 5.4670mWh | 5.2472mWh |
| 平均 | 5.6362mWh | 6.2177mWh | 5.4583mWh | 5.2400mWh |

未接続の状態からセンサーを接続すると+13.10mW、さらに接続している状態から遠くを測定すると+10.67mW、近くを測定すると+45.56mW消費電力が上がることがわかります。

遠くの測定と近くの測定でここまで変化するのは意外でした。消費電力の計算上は、近くを測定する想定の方が良いかも知れません。

## -4. GPS

GPSによる測位を行うことができます。

2秒ごとに1分間GPSでデータを取得します。比較としてデータ取得動作をしていない状態、電源OFFの状態も測定します。

```
#include <WioLTEforArduino.h>

#define INTERVAL    (2000)

WioLTE Wio;
unsigned long uptime;
unsigned int count = 0;

void setup()
{
  delay(200);
  GpsBegin(&SerialUART);
  Wio.Init();
  Wio.PowerSupplyGrove(true);
  delay(500);

  SerialUSB.println("### Setup completed.");
}

void loop()
{
  count++;
  if (count <= 45){
    const char* data = GpsRead();
```

```
    const char* data = GpsRead();

    if (count == 45){
      if (data != NULL && strncmp(data, "$GPGGA,", 7) == 0) {
        SerialUSB.println(data);
      }
      SerialUSB.println("Finish Measure GPS");
    }
  } else if (count <= 90){
    if (count == 90){
      SerialUSB.println("Finish no operation state");
      Wio.PowerSupplyGrove(false);
    }
  } else {
    if (count == 135){
      count = 0;
      Wio.PowerSupplyGrove(true);
      delay(500);
      SerialUSB.println("Finish no power state");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}

// 省略：GpsBegin
// 省略：GpsRead
```

| 項目 | GPS取得×30回 | 取得せず | 電源OFF |
|---|---|---|---|
| 1回目 | 7.3327mWh | 7.1313mWh | 5.1748mWh |
| 2回目 | 7.3378mWh | 7.2386mWh | 5.1792mWh |
| 3回目 | 7.3679mWh | 7.1641mWh | 5.1271mWh |
| 平均 | 7.3461mWh | 7.1780mWh | 5.1603mWh |

GPSモジュールに電源を入れていることによる電力消費は大きく、電源を入れているだけで消費電力にして121.1mWもの違いが出ることがわかります。必要がなければ極力GPSの電源はOFFにしていた方がよさそうです。

GPSを使用することによる消費は１回あたり5.603μWhとなります。

## -5. スイッチ

スイッチが押されているかどうかを取得することができます。

100ミリ秒ごとにスイッチの状態と取得します。スイッチを押していない状態と押している状態の両方を取得し、スイッチの状態を取得していない時と比較します。

```
#include <WioLTEforArduino.h>

#define BUTTON_PIN  (WIOLTE_D38)
#define INTERVAL    (100)

unsigned long uptime;
unsigned int count = 0;

void setup()
{
  pinMode(BUTTON_PIN, INPUT);
}

void loop()
{
```

```
{

  count++;
  if (count <= 900){
    int buttonState = digitalRead(BUTTON_PIN);
    if (count == 900){
      SerialUSB.println(buttonState);
      SerialUSB.println("Finish Measure state without push");
    }
  } else if (count <= 1800){
    int buttonState = digitalRead(BUTTON_PIN);
    if (count == 1800){
      SerialUSB.println(buttonState);
      SerialUSB.println("Finish Measure state with push");
    }
  } else {
    if (count == 2700){
      count = 0;
      SerialUSB.println("Finish no operation state");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}
```

| 項目 | スイッチOFF取得(600回) | スイッチON取得(600回) | 取得せず |
|---|---|---|---|
| 1回目 | 5.1317mWh | 5.3518mWh | 5.1344mWh |
| 2回目 | 5.1409mWh | 5.3772mWh | 5.1464mWh |
| 3回目 | 5.1700mWh | 5.3664mWh | 5.1596mWh |
| 平均 | 5.1475mWh | 5.3651mWh | 5.1468mWh |

スイッチの状態を取得するかどうかは消費電力にはほぼ影響しません。
一方スイッチを押しているかは大きく影響し、1分間の消費電力量で0.2176mWh、消費電力にして13.06mWもの違いが生じている事がわかります。（スイッチの状態を取得する/しないに関わらず）スイッチはOFF状態を基本と考えた方がいいですね。

## -6. 磁気スイッチ

スイッチとほぼ同じですが、こちらは磁石を近づけるとスイッチを押した状態となります。
基本的にはスイッチの測定と同じです。100ミリ秒ごとに磁気スイッチの状態を取得します。磁気スイッチの近くに磁石がない状態とある状態の両方を取得し、磁気スイッチの状態を取得していない時と比較します。

```
#include <WioLTEforArduino.h>

#define MAGNETIC_SWITCH_PIN (WIOLTE_D38)
#define INTERVAL    (100)
#define NO_OPERATION_TIMESPAN (90000)

unsigned long uptime;
unsigned int count = 0;

void setup()
{
  pinMode(MAGNETIC_SWITCH_PIN, INPUT);
}

void loop()
{
  count++;
```

```
  if (count <= 900){
    int switchState = digitalRead(MAGNETIC_SWITCH_PIN);
    if (count == 900){
      SerialUSB.println(switchState);
      SerialUSB.println("Finish Measure state without magnet");
    }
  } else if (count <= 1800){
    int switchState = digitalRead(MAGNETIC_SWITCH_PIN);
    if (count == 1800){
      SerialUSB.println(switchState);
      SerialUSB.println("Finish Measure state with magnet");
    }
  } else {
    if (count == 2700){
      count = 0;
      SerialUSB.println("Finish no operation state");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}
```

| 項目 | 磁石なし取得(600回) | 磁石あり取得(600回) | 取得せず |
|---|---|---|---|
| 1回目 | 5.2619mWh | 5.2562mWh | 5.2451mWh |
| 2回目 | 5.2406mWh | 5.2529mWh | 5.2641mWh |
| 3回目 | 5.2706mWh | 5.2492mWh | 5.2765mWh |
| 平均 | 5.2577mWh | 5.2527mWh | 5.2619mWh |

消費電力はほぼ変化なしですね。こちらは通常のプッシュスイッチと違い、磁石が近くにある場合と無い場合で、消費電力による変化はありませんでした。

## -7. ブザー

ブザーを鳴らす事ができます。

2秒ごとに100ミリ秒、もしくは1秒のブザーを鳴動させます。比較のため、鳴動させない期間も測定します。

```
#include <WioLTEforArduino.h>

#define BUZZER_PIN      (WIOLTE_D38)
#define BUZZER_ON_SHORT  (100)
#define BUZZER_ON_LONG  (1000)
#define INTERVAL (2000)

unsigned long uptime;
unsigned int count = 0;

void setup()
{
  pinMode(BUZZER_PIN, OUTPUT);
  delay(500);
}

void loop()
{
  count++;
  if (count <= 45){
    digitalWrite(BUZZER_PIN, HIGH);
    delay(BUZZER_ON_SHORT);
    digitalWrite(BUZZER_PIN, LOW);
```

```
  }
  else if (count <= 90){
    digitalWrite(BUZZER_PIN, HIGH);
    delay(BUZZER_ON_LONG);
    digitalWrite(BUZZER_PIN, LOW);
  }
  else if (count == 135){
    count = 0;
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}
```

| 項目 | ブザー(100ミリ秒)×30回 | ブザー(1秒)×30回 | 鳴動なし |
|---|---|---|---|
| 1回目 | 5.2032mWh | 5.7605mWh | 5.1456mWh |
| 2回目 | 5.2090mWh | 5.7712mWh | 5.1594mWh |
| 3回目 | 5.2130mWh | 5.7658mWh | 5.1566mWh |
| 平均 | 5.2084mWh | 5.7658mWh | 5.1538mWh |

ブザー(100ミリ秒)は1回あたり1.82μWh、ブザー(1秒)は1回あたり20.40μWhになることがわかります。

ブザーはかなり電力を消費することがわかります。また鳴動時間を長くするとそれだけ消費電力も大きくなります。必要最小限の長さにする必要がありそうです。

## LTEモジュール

Wio LTEの特長は、何と言ってもこの小さい基板の中にLTEモジュールとSIMソケットがすでに搭載されていることです。これによりセンサーで測定した値をクラウドに送信する、といったことが非常に容易になっています。

また、SORACOMのサービスと組み合わせると、負荷の高い暗号化や認証などの処理をSORACOMにオフロードすることができ、モジュールからは単にTCPやUDPで送信・受信するだけで良くなります。

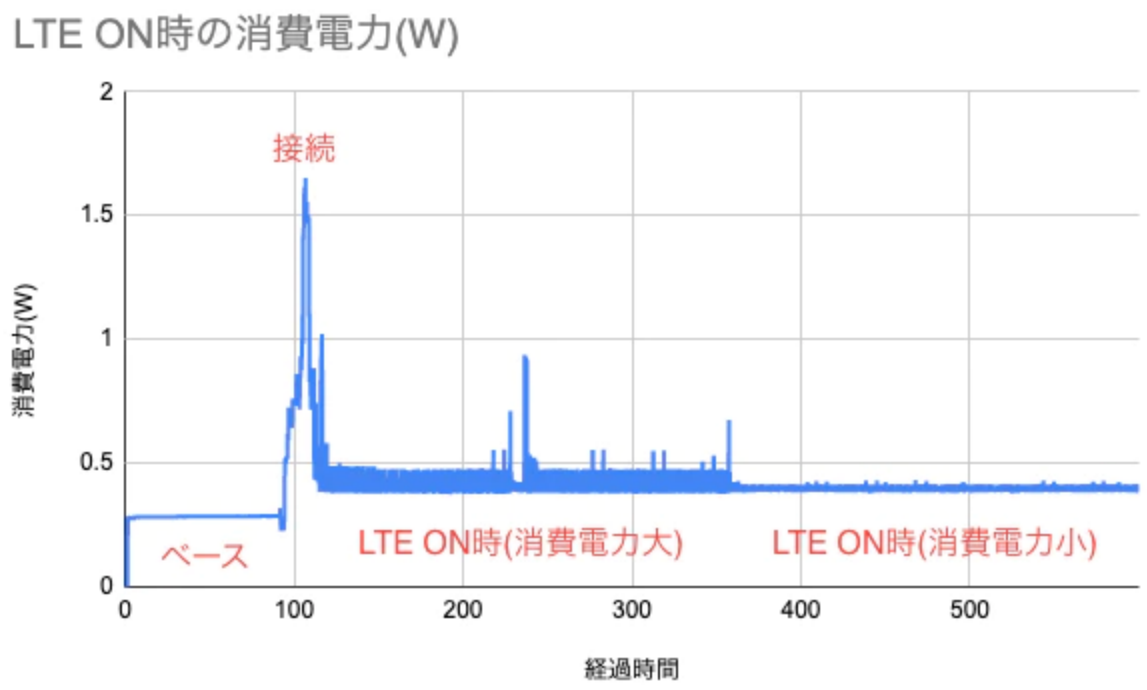
ここからはSORACOMに接続し、SORACOM HarvestにUDPでデータ送信する場合の消費電力を測定していきます。

### -1. LTEの電源をONにする

```
#include <WioLTEforArduino.h>
#define INTERVAL  (5000)
WioLTE Wio;

void setup() {
  delay(200);
  Wio.Init();
  SerialUSB.println("Finish Wio Init");
  delay(90000); // ベースの電力量を測るための休止期間
  SerialUSB.println("Connect");
  Wio.PowerSupplyLTE(true);
  delay(500);
  if (!Wio.TurnOnOrReset()) {
    SerialUSB.println("### ERROR! ###");
    return;
  }
```

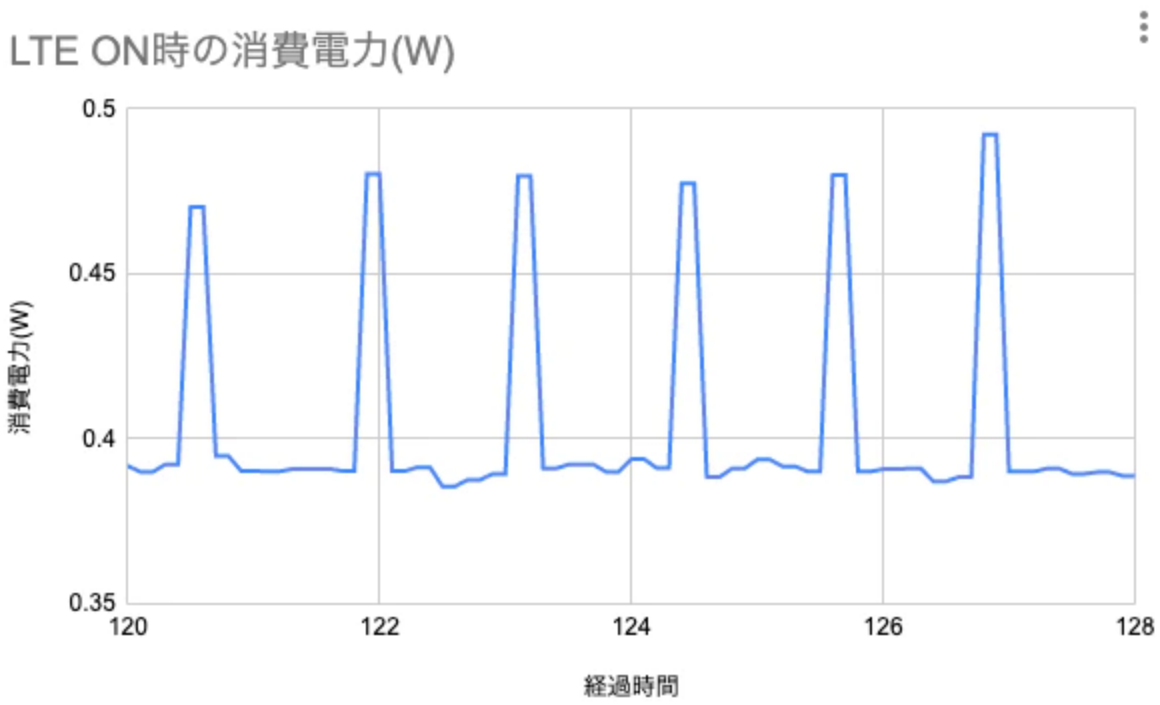
```
  SerialUSB.println("Finish Wio Power Supply and Turn ON");
}

void loop() {
  delay(INTERVAL);
}
```

LTEをONにする際の電力推移は以下のようになります。

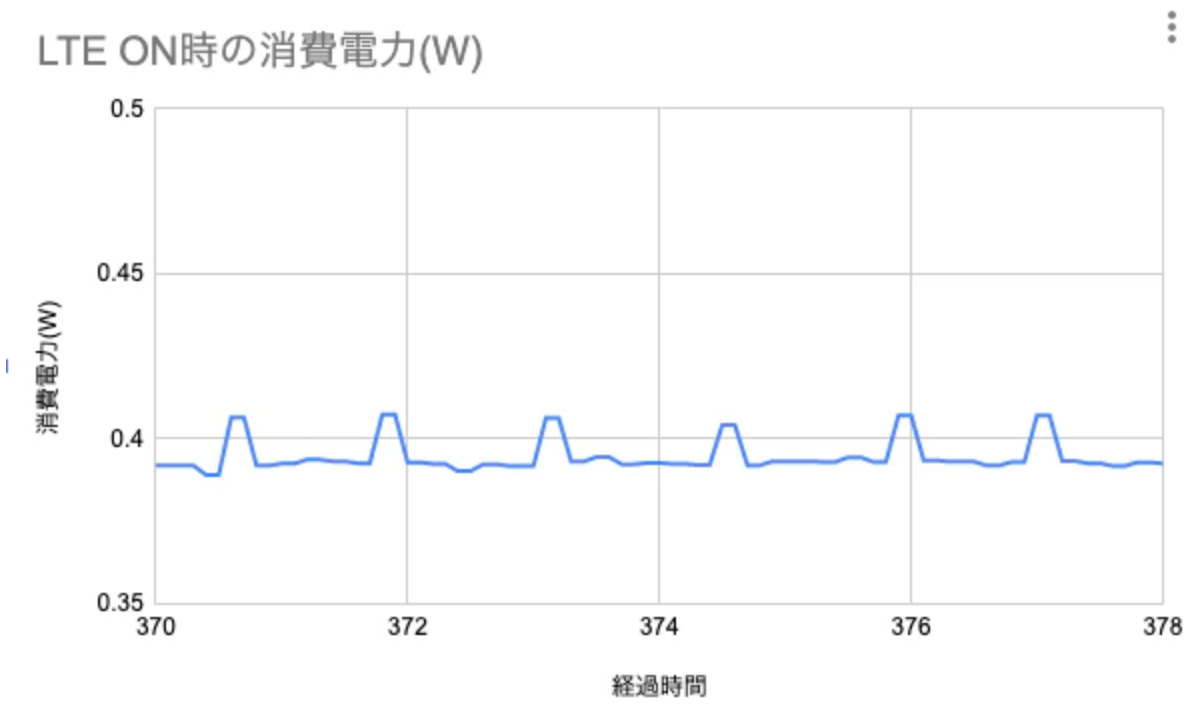



LTEをONにした後に大きく消費し、その後一定時間は大きめの消費、しばらくしてから小さめの消費になることがわかります。なぜしばらくしたら小さめの消費になるのかは不明です。

消費電力が大きい時のグラフの拡大



消費電力が小さい時のグラフの拡大



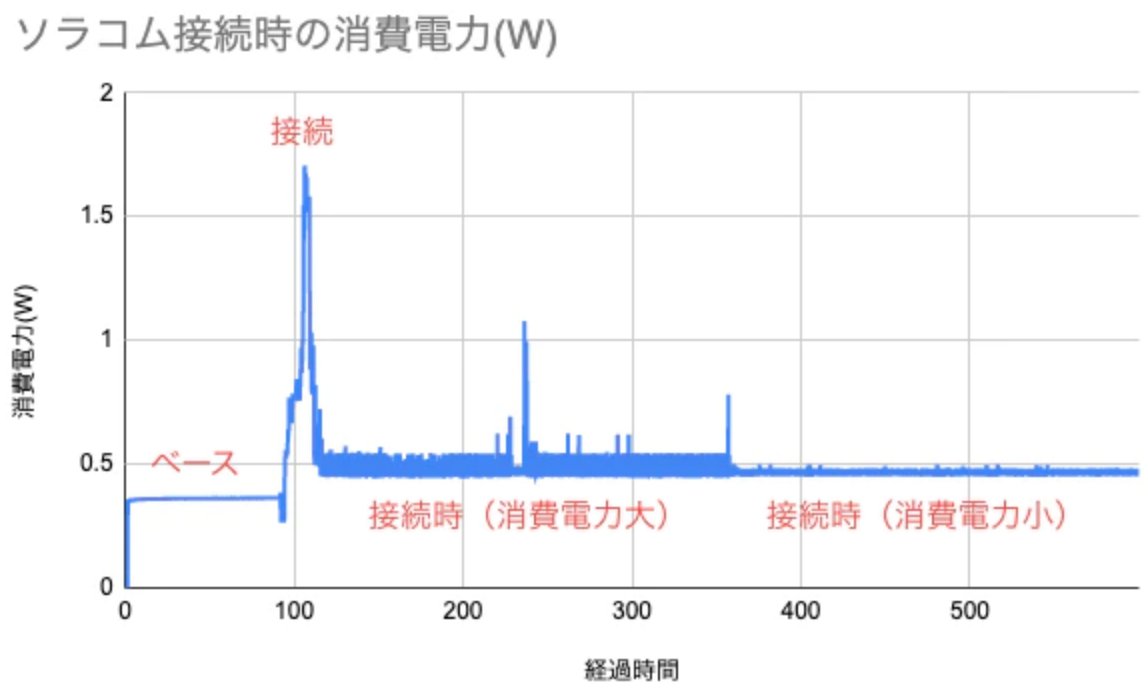

ベースはほぼ変わりませんが、一定時間ごとのピークがかなり小さくなっています。この一定時間のピークはDRXと呼ばれるLTEの待ち受け確認のための動作です。（省電力の規格であるLTE-MではこのDRXの間隔を大幅に延ばすことができるとのことです）

| 項目 | ベース | 接続時(消費電力大) | 接続時（消費電力小） |
|---|---|---|---|
| 1回目 | 4.7197mWh | 6.7303mWh | 6.6021mWh |
| 2回目 | 4.6870mWh | 6.7931mWh | 6.6631mWh |
| 3回目 | 4.7272mWh | 6.7701mWh | 6.6153mWh |
| 平均 | 4.7113mWh | 6.7645mWh | 6.6268mWh |

LTEをONにするとOFFの時に比べ消費電力は123.2mW上がり、
しばらく何もしないと消費電力は8.26mW下がります。

## -2. ソラコムに接続する

```
#include <WioLTEforArduino.h>
#define BASE_TIMESPAN (1 * 60 * 1000)
#define INTERVAL  (5 * 1000)
WioLTE Wio;

void setup() {
  delay(200);
  Wio.Init();
  delay(BASE_TIMESPAN); // ベースの電力量を測るための休止期間
  Wio.PowerSupplyLTE(true);
  delay(500);
  if (!Wio.TurnOnOrReset()) {
    SerialUSB.println("### ERROR! ###");
    return;
  }
  delay(1 * 60 * 1000); // 接続前後の変化をみるための休止期間
  if (!Wio.Activate("soracom.io", "sora", "sora")) {
    SerialUSB.println("### ERROR! ###");
    return;
  }
}

void loop() {
  delay(INTERVAL);
}
```




| 項目 | ベース | 接続時(消費電力大) | 接続時（消費電力小） |
|---|---|---|---|
| 1回目 | 6.0926mWh | 7.8797mWh | 7.7816mWh |
| 2回目 | 6.1471mWh | 7.9174mWh | 7.8195mWh |
| 3回目 | 6.1411mWh | 7.9302mWh | 7.8097mWh |
| 平均 | 6.1269mWh | 7.9091mWh | 7.8036mWh |

接続しているとしていない時に比べ消費電力は106.9mW上がり、
しばらく通信していないと消費電力は6.32mW下がります。

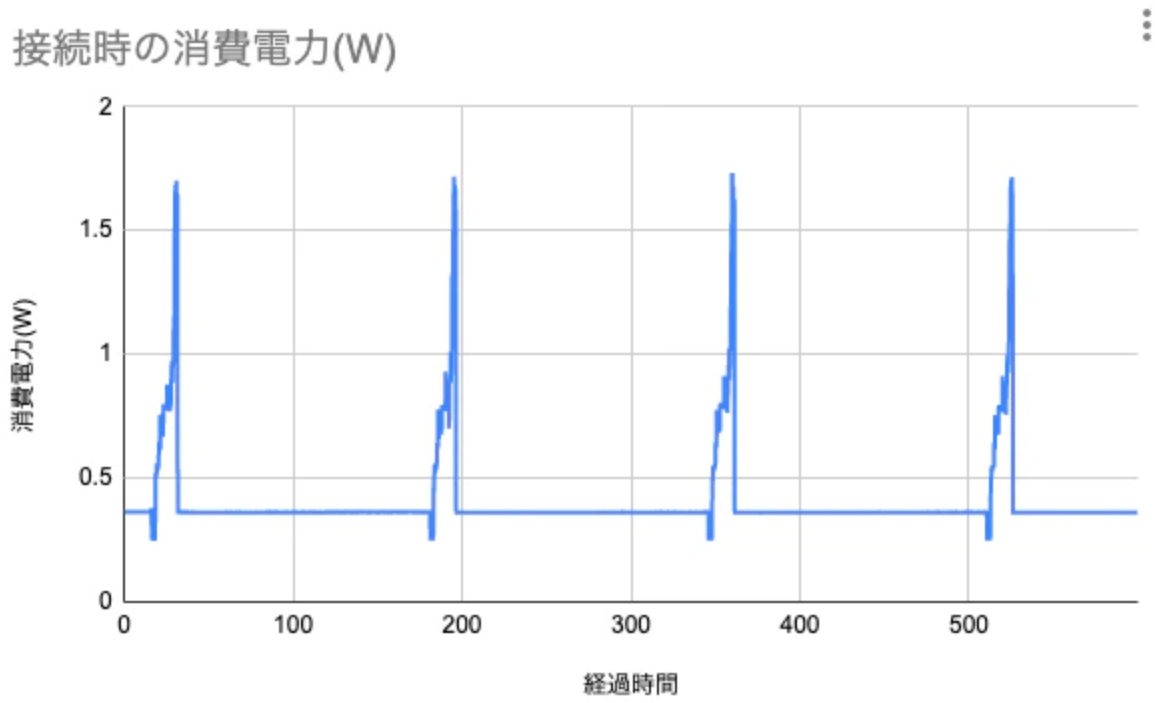
LTEをONにする時よりも、未接続時との違いが小さくなるのは意外でした。
ですが、**Wio.Activate**がプログラムに入るだけでなぜかベースの消費電力が84.94mWも上がります。接続などをしていなくても。ビルドのサイズがActivateなしが50992byteなのに対し、Activateありが53504byteに上がっているため、これが影響で消費電力が上がっているのかも知れません。このように使用する機能によってベースの消費電力があがってしまう、ということは心にとめておくべきかも知れません。

## -3. 接続

LTE電源供給、接続、切断、電源供給停止を繰り返し、接続していない時との差から、接続処理1回あたりの消費電力量を割り出します。

```
#include <WioLTEforArduino.h>
WioLTE Wio;
unsigned long uptime;

void setup() {
  delay(200);
  Wio.Init();
}

void loop() {
  SerialUSB.println("Connect start after 10 seconds");
  // ここから測定開始
  delay(10000);
  Wio.PowerSupplyLTE(true);
  delay(500);
  if (!Wio.TurnOnOrReset()) {
    SerialUSB.println("### Turn ON ERROR! ###");
    return;
  }
  if (!Wio.Activate("soracom.io", "sora", "sora")) {
```

```
    SerialUSB.println("### Activate ERROR! ###");
    return;
  }
  if (!Wio.Deactivate()) {
    SerialUSB.println("### Deactivate ERROR! ###");
    return;
  }
  Wio.PowerSupplyLTE(false);
  // 測定が終わったら、ベース消費電力の測定
  delay(140000);
}
```




| 項目 | 接続あり | 接続なし |
|---|---|---|
| 1回目 | 7.899mWh | 6.052mWh |
| 2回目 | 7.812mWh | 6.039mWh |
| 3回目 | 7.805mWh | 6.054mWh |
| 4回目 | 7.899mWh | 6.041mWh |
| 5回目 | 7.875mWh | 6.049mWh |
| 平均 | 7.858mWh | 6.047mWh |

接続/切断処理1回あたりの消費電力量は1.811mWhとなります。

-2.ソラコムに接続する、の時の結果から、接続している時としていない時では消費電力に106.9mWの差があります。

1.811(mWh) / 106.9(mW) ≃ 0.01694(h) ≃ 61(s)

となりますので、接続１回あたりの電力量は接続していることによる増加分のおおよそ1分にあたります。
従って、１分以内で送信を繰り返す場合などは接続を維持した方がよく、逆に１分を越える間隔で送信する場合などは都度接続・切断をした方が良いと考えられます。（接続失敗しての再接続などは考慮しないとする）

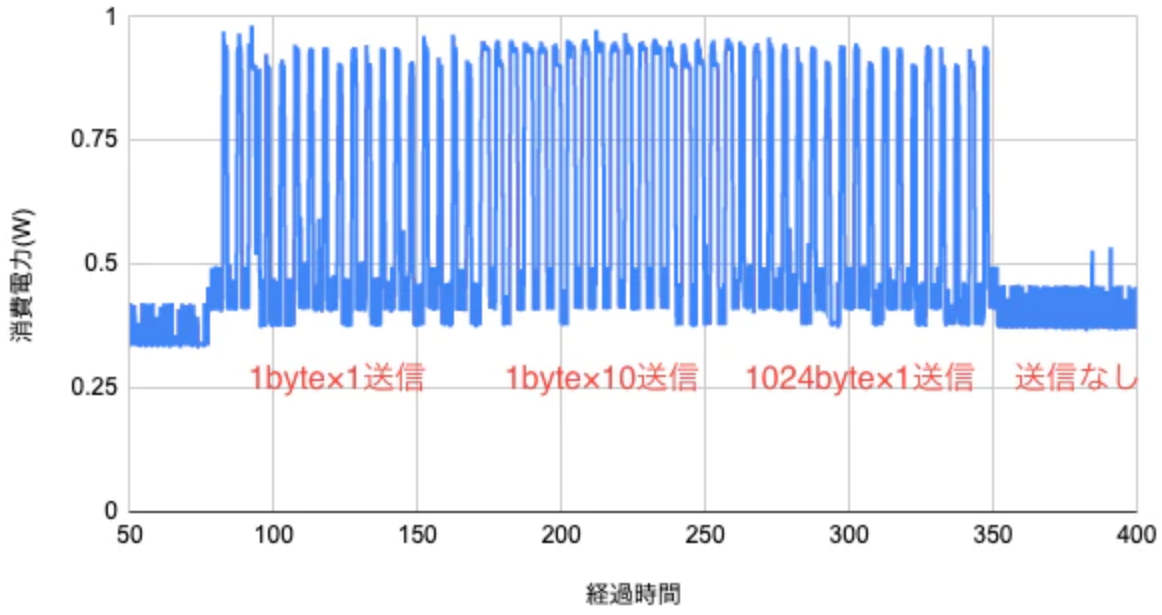
リアルタイム送受信が必要ない多くのケースでは、都度接続・切断でもよいような気がします。

## -4. SORACOM HarvestへのUDP送信

色々な送信の仕方がありますが、今回は5秒ごとに1byteで1回送信した場合、1byteを10回送信した場合、1024byteを1回送信した場合をそれぞれ測定します。

```
#include <WioLTEforArduino.h>
#define INTERVAL  (5000)
#define RECEIVE_TIMEOUT (10000)
WioLTE Wio;
unsigned long uptime;
char shortData[1024];
char longData[1025];
char recvData[1024];
unsigned int count = 0;

void setup() {
  delay(200);
  strcpy(shortData, "1");
  memset(longData, '1', 1024);
  longData[1024] = 0;
  Wio.Init();

  Wio.PowerSupplyLTE(true);
  delay(500);
  if (!Wio.TurnOnOrReset()) {
    SerialUSB.println("### Turn ON ERROR! ###");
    return;
  }
  if (!Wio.Activate("soracom.io", "sora", "sora")) {
    SerialUSB.println("### Activate ERROR! ###");
    return;
  }
  SerialUSB.println("Connected");
}

void loop() {
  count++;
  if (count <= 18){
    uploadHarvest(shortData);
    if (count == 18){
      SerialUSB.println("Finish upload single short data");
    }
  } else if (count <= 36){
    for (unsigned int i = 0; i < 10; i++){
      uploadHarvest(shortData);
    }
    if (count  == 36){
      SerialUSB.println("Finish upload many short data");
    }
  } else if (count <= 54){
    uploadHarvest(longData);
    if (count == 54){
      SerialUSB.println("Finish upload single long data");
    }
  } else if (count == 72){
    if (count == 72){
      count = 0;
      SerialUSB.println("Finish base time");
    }
  }

  uptime = millis();
  delay(INTERVAL - (uptime % INTERVAL));
}

void uploadHarvest(char *data) {
  int connectId;
  connectId = Wio.SocketOpen("harvest.soracom.io", 8514, WIOLTE_UDP);
  if (connectId < 0) {
    SerialUSB.println("### Open ERROR! ###");
    return;
  }
  if (!Wio.SocketSend(connectId, data)) {
    SerialUSB.println("### Send ERROR! ###");
```

```
    goto err_close;
  }
  int length;
  length = Wio.SocketReceive(connectId, recvData, sizeof (recvData), RECEIVE_TIMEOUT);
  if (length < 0) {
    SerialUSB.println("### Recieve ERROR! ###");
    goto err_close;
  }
  if (length == 0) {
    SerialUSB.println("### RECEIVE TIMEOUT! ###");
    goto err_close;
  }
err_close:
  if (!Wio.SocketClose(connectId)) {
    SerialUSB.println("### Close ERROR! ###");
    return;
  }
}
```




| 項目 | ベース | 1byte×1 | 1byte×10 | 1024byte×1 |
|---|---|---|---|---|
| 1回目 | 5.913mWh | 9.555mWh | 11.423mWh | 9.429mWh |
| 2回目 | 5.877mWh | 9.552mWh | 11.427mWh | 9.404mWh |
| 3回目 | 5.870mWh | 9.492mWh | 11.427mWh | 9.477mWh |
| 平均 | 5.886mWh | 9.533mWh | 11.425mWh | 9.436mWh |

１回あたりの消費電力量は以下のようになります。

- 1byte×1回：0.304mWh
- 1byte×10回：0.462mWh
- 1024byte×1回：0.296mWh

UDP送信が1パケットで収まる分には消費電力はほぼ変わりませんが、送信バイト数が少なくともパケット数が多くなると消費電力は大きくなるようです。単純にパケット数倍になるわけではないようですが、データは可能な限り1パケットにまとめるのがよいでしょう。

## Wio Extension - RTCによる電源遮断

ここまでみてきた中で、大きな問題はそもそも動作しているだけでおおよそ300mW程度は消費し続けていることです。

代表的なモバイルバッテリー(2019/12/22時点でAmazonのモバイルバッテリー部門1位のAnker PowerCore 10000を例にします)は電池容量が36Whなので、これを使用すると考えると120時間 = 5日しか持ちません。電池で長時間持つとは言えないです。

このような場合のために、電源を遮断してRTC(リアルタイムクロック)にて復帰するWio Extension RTCという拡張ボードが用意されていますので、それを使用した場合の消費電力を測定してみました。

ただし残念ながらこの測定はあまり正確にはできませんでした。USBケーブル自体が何らかの電力を消費しているのか、ケーブルをWio Extension RTCに接続していない状態でも52μA程度の電流が流れてしまっているためです。そのため、同じケーブルでボードに接続している場合としていない場合で1時間ずつ積算電力量を計算し、その差を消費電力量とします。

| ケーブルのみ | RTC接続 | 差 |
|---|---|---|
| 0.01294mWh | 0.01398mWh | 0.00104mWh |

Wio Extension - RTCを接続することによる増加分は0.00104mWhであり、電圧が5Vちょうどとすると0.208μAです。消費電流が1μA以下という記載の裏付けが取れました。

36Whのバッテリーであれば約4030年動作できる程度の消費電力なので、事実上RTCによる消費電力は無視して良いでしょう。ただし電流が小さすぎるとモバイルバッテリーが出力をOFFにしてしまう場合があるようなので、小電流供給に対応したモバイルバッテリーを使う必要があるとのことです。(こういうのでしょうか？)

## 実際の動作に近い場合

基本的な方針としては、立ち上がったらすぐに測定してデータをアップロードして、次回立ち上がりの予定をしてすぐに電源を落とす、という動作をすることになります。たとえば以下は１時間に１回だけ温湿度を測定してアップロードする例です。

```
#include <WioLTEforArduino.h>
#include "WioRTC.h"
#define BOOT_INTERVAL   (3600)
#define SENSOR_PIN    (WIOLTE_D38)

WioLTE Wio;
WioRTC Rtc;

void setup()
{
  delay(200);
  Wio.Init();
  Wio.PowerSupplyGrove(true);
  delay(500);
  Wire.begin();
  Rtc.begin();
}

void loop()
{
  byte data[4];
  TemperatureAndHumidityBegin(SENSOR_PIN);
  if (!TemperatureAndHumidityRead(data)) {
    SerialUSB.println("ERROR!");
    return;
  }

  Wio.PowerSupplyLTE(true);
  delay(500);
```

```
  if (!Wio.TurnOnOrReset()) {
    SerialUSB.println("### Turn ON ERROR! ###");
    return;
  }
  if (!Wio.Activate("soracom.io", "sora", "sora")) {
    SerialUSB.println("### Activate ERROR! ###");
    return;
  }

  int connectId;
  connectId = Wio.SocketOpen("harvest.soracom.io", 8514, WIOLTE_UDP);
  if (connectId < 0) {
    SerialUSB.println("### Open ERROR! ###");
    return;
  }
  if (!Wio.SocketSend(connectId, data, 4)) {
    SerialUSB.println("### Send ERROR! ###");
    return;
  }
  int length;
  char recvData[1024];
  length = Wio.SocketReceive(connectId, recvData, sizeof (recvData), 5000);
  if (length < 0) {
    SerialUSB.println("### Recieve ERROR! ###");
    return;
  }
  if (length == 0) {
    SerialUSB.println("### RECEIVE TIMEOUT! ###");
    return;
  }

  Rtc.SetWakeupPeriod(BOOT_INTERVAL);
  Rtc.Shutdown();
  while (1) {}
}

// 省略：TemperatureAndHumidityBegin
// 省略：TemperatureAndHumidityRead
```

この場合、１回あたりの消費電力量は3.302mWhとなりました。36Whのモバイルバッテリーであれば、約454日稼働できることになります。1年保たせられれば、上々と言えるかと思います。もっと測定間隔を短くしたい場合は、１分に１回立ち上がり測定してデータをRTCボードのEEPROMに保存し、データが1時間分貯まったらまとめてアップロードする、などが考えられます。（今回は測定していませんが）

## まとめ

| 項目 | 消費電力 | 1回あたりの消費電力量 |
|---|---|---|
| 温湿度測定 | - | 13.24μWh |
| 加速度センサー接続 | 1.476mW | - |
| 加速度測定(100ms間隔) | ほぼ0 | - |
| 超音波距離センサー電源ON | 13.10mW | - |
| 超音波距離測定(100ms間隔：遠く) | 10.67mW | - |
| 超音波距離測定(100ms間隔：近く) | 45.56mW | - |
| GPS電源ON | 121.1mW | - |
| GPS使用 | - | 5.603μWh |
| スイッチOFF | ほぼ0 | - |

| 項目 | 消費電力 | 1回あたりの消費電力量 |
|---|---|---|
| | | |
| スイッチON | 13.06mW | - |
| 磁気スイッチOFF | ほぼ0 | - |
| 磁気スイッチON | ほぼ0 | - |
| ブザー100ms | - | 1.82μWh |
| ブザー1s | - | 20.40μWh |
| LTE ON | 123.2mW | - |
| ソラコム接続 | 106.9mW | - |
| LTE接続処理 | - | 1.811mWh |
| UDP 1024byte 送信 | - | 0.296mWh |
| Wio Extension RTC | 0.00102mW | - |

正直一番消費が大きいのはボードが動作していることそのものなのですが、そちらは状況により一概には測定できませんでした。何か定式化できれば良いんですが。

RTCと組み合わせれば、電池で１年以上、という使い方もできそうなので、何かデバイスを思いついたらやってみようかな。

以上です。



@1stship

フォロー

## コメント

この記事にコメントはありません。

あなたもコメントしてみませんか :)

ユーザ登録

すでにアカウントを持っている方はログイン