

IoTシステム工学レポート

4135 舩屋 耀大

ステップ1 ATOM Matrixの加速度センサを使い、Wi-Fiに接続する

目標

- エッジデバイス(ATOM Matrix)の開発環境の確認
 - arduino-cliを使ったATOM Matrix(ESP32)の開発
 - マイコンのコードをクラス設計して開発する
- ATOM Matrixの加速度センサを使う

ビルド方法の確認

```
PS C:\Users\r04i36\source\repos> cd \Users\r04i36\Documents\IoTシステム工学\実験
PS C:\Users\r04i36\Documents\IoTシステム工学\実験> cd prototype-system-masuyayout
PS C:\Users\r04i36\Documents\IoTシステム工学\実験\prototype-system-masuyayout> cd edge
PS C:\Users\r04i36\Documents\IoTシステム工学\実験\prototype-system-masuyayout\edge> rmake board

Microsoft(R) Program Maintenance Utility Version 14.43.34810.0
Copyright (C) Microsoft Corporation. All rights reserved.

"C:\Program Files\arduino-ide\resources\app\lib\backend\resources\arduino-cli.exe" --config-file c:\Users\r04i36\.arduinoIDE\arduino-cli.yaml board list
シリアルポート プロトコル タイプ Board Name FQBN Core
COM1 serial Serial Port Unknown
COM3 serial Serial Port (USB) Unknown
```

```
PS C:\Users\r04i36\Documents\IoTシステム工学\実験\prototype-system-masuyayout\edge> rmake build

Microsoft(R) Program Maintenance Utility Version 14.43.34810.0
Copyright (C) Microsoft Corporation. All rights reserved.

docker run -v ./work -it --rm arduino-cli sh -c "cd /work && arduino-cli compile -b esp32:esp32:m5stack-atom edge_device --output-dir edge_device/bin/"
Sketch uses 231085 bytes (17%) of program storage space. Maximum is 1310720 bytes.
Global variables use 20968 bytes (6%) of dynamic memory, leaving 306712 bytes for local variables. Maximum is 327680 bytes.
```

```
PS C:\Users\r04i36\Documents\IoTシステム工学\実験\prototype-system-masuyayout\edge> rmake write

Microsoft(R) Program Maintenance Utility Version 14.43.34810.0
Copyright (C) Microsoft Corporation. All rights reserved.

"C:\Program Files\arduino-ide\resources\app\lib\backend\resources\arduino-cli.exe" --config-file c:\Users\r04i36\.arduinoIDE\arduino-cli.yaml upload -p COM3 -b esp32:esp32:m5stack-atom --input-dir edge_device\bin
esptool.py v4.5.1
Serial port COM3
Connecting...
Chip is ESP32-PICO-D4 (revision v1.0)
Features: WiFi, BT, Dual Core, 240MHz, Embedded Flash, VRef calibration in efuse, Coding Scheme None
Crystal is 40MHz
MAC: e8:9f:6d:08:97:d8
Uploading stub...
Running stub...
Stub running...
Changing baud rate to 1500000
Changed.
Configuring flash size...
Flash will be erased from 0x00001000 to 0x00005fff...
Flash will be erased from 0x00008000 to 0x00008fff...
Flash will be erased from 0x0000e000 to 0x0000ffff...
Flash will be erased from 0x00010000 to 0x00040fff...
Compressed 17568 bytes to 12205...
Writing at 0x00001000... (100 %)
Wrote 17568 bytes (12205 compressed) at 0x00001000 in 0.4 seconds (effective 343.7 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 146...
Writing at 0x00008000... (100 %)
Wrote 3072 bytes (146 compressed) at 0x00008000 in 0.1 seconds (effective 327.4 kbit/s)...
Hash of data verified.
Compressed 8192 bytes to 47...
Writing at 0x0000e000... (100 %)
Wrote 8192 bytes (47 compressed) at 0x0000e000 in 0.1 seconds (effective 531.6 kbit/s)...
Hash of data verified.
Compressed 231456 bytes to 128214...
Writing at 0x00010000... (12 %)
Writing at 0x00010073... (25 %)
Writing at 0x00023833... (37 %)
Writing at 0x00028b5c... (50 %)
Writing at 0x0002e01f... (62 %)
Writing at 0x00036539... (75 %)
Writing at 0x0003e6e3... (87 %)
Writing at 0x00043cb3... (100 %)
Wrote 231456 bytes (128214 compressed) at 0x00010000 in 2.4 seconds (effective 773.9 kbit/s)...
Hash of data verified.
Leaving...
Hard resetting via RTS pin...
New upload port: COM3 (serial)
```

[設計]テスト項目

1. Atom Matrixを動かして、加速度が変化するか確認する
2. 加速度をJSON形式で表示する

[実装1]Atom Matrixの加速度センサを使う

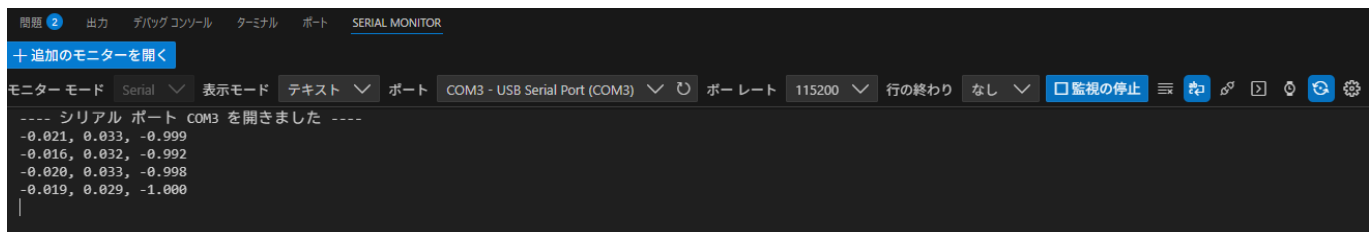
```
PS C:\Users\r04i36\Documents\IoTシステム工学\実験\prototype-system-masuyayout\edge> nmake build

Microsoft(R) Program Maintenance Utility Version 14.43.34810.0
Copyright (C) Microsoft Corporation. All rights reserved.

    docker run -v ./work -it --rm arduino-cli sh -c "cd /work && arduino-cli compile -b esp32:esp32:m5stack-atom edge_device --output-dir edge_device/bin/"
Sketch uses 318081 bytes (24%) of program storage space. Maximum is 1310720 bytes.
Global variables use 22760 bytes (6%) of dynamic memory, leaving 304920 bytes for local variables. Maximum is 327680 bytes.
```

[テスト1] シリアルモニターでデバッグし、加速度センサが動作するか確認する

1. VS Codeに"Serial Monitor"を追加する
2. 「シリアルモニター」タブのポートをATOM Matrixが接続されているポートに指定し、ボーレートを entorypoint.cppで設定した「115200」に変更し、「監視の開始」をクリックする
3. シリアルコンソールに加速度が10秒ごとに表示される



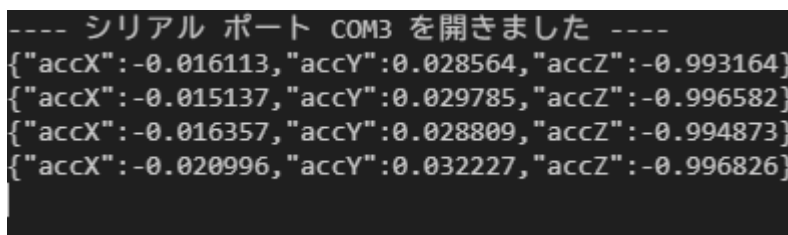
The screenshot shows the VS Code Serial Monitor interface. The 'Serial' tab is selected, and the port is set to 'COM3 - USB Serial Port (COM3)'. The baud rate is set to '115200'. The 'Monitor' button is clicked, and the output shows acceleration data being received from the ATOM Matrix.

```
---- シリアル ポート COM3 を開きました ----
-0.021, 0.033, -0.999
-0.016, 0.032, -0.992
-0.020, 0.033, -0.998
-0.019, 0.029, -1.000
```

[実装2]加速度センサのデータをJSON形式に変換する

[テスト2] 加速度がJSONで表示されるか確認する

- Atom Matrixを動かしたときに、加速度がJSONで表示される



The screenshot shows the VS Code Serial Monitor interface with the output displaying acceleration data in JSON format.

```
---- シリアル ポート COM3 を開きました ----
{"accX": -0.016113, "accY": 0.028564, "accZ": -0.993164}
{"accX": -0.015137, "accY": 0.029785, "accZ": -0.996582}
{"accX": -0.016357, "accY": 0.028809, "accZ": -0.994873}
{"accX": -0.020996, "accY": 0.032227, "accZ": -0.996826}
```

ステップ2 ATOM MatrixをWi-Fiに接続する

目標

- ATOM MatrixをWi-Fiでネットワークに接続し、疎通確認する
 - Wi-Fi APへの接続
 - pingによる疎通確認

[設計]テスト項目

- Wi-Fi APへの接続を確認する
 - Wi-Fi APのMACアドレスが得られている
 - DHCPサーバからIPアドレスが振られる
- 疎通確認する

- 同一セグメント内の別マシンにpingが通る

[実装]ATOM MatrixをWi-Fiに繋いで実験室PCにpingを送る

[テスト]テスト項目の確認

- Wi-Fi APのMACアドレスがシリアルコンソールに表示されるか？
- Wi-Fiに接続し、ATOM Matrixに割り振られたIPアドレスが表示されるか？
- pingに対し、Successと表示されるか？

```
---- シリアル ポート COM3 を開きました ----
Success
{"accX":-0.00708,"accY":-0.018311,"accZ":-1.000488}
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 188777542, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1184
load:0x40078000,len:13260
load:0x40080400,len:3028
entry 0x400805e4
M5Atom initializing...OK
19
E8:9F:6D:08:97:D8
WiFi connected
IP address:
172.16.98.206
```

ステップ2 MQTTブローカーを立ち上げる

目標

- MQTTブローカーを起動する
 - dockerを使ったサーバの起動、開発環境の整備
 - MQTTの動作確認

[設計]テスト項目

- mqttサーバが起動し、ローカル環境でpublish、subscribeができる

[設定] mosquittoのbrokerを立ち上げる(Dockerコンテナ)、brokerに対して、subscribe、publishして動作確認する

```
/usr/src/app # mosquitto_passwd -c -b /mosquitto/mosquitto_passwd.txt masuyayouta masuya038
```

```
codespace →/workspaces/prototype-system-masuyayout/gateway (main) $ docker compose up -d
[+] Running 2/2
  ✓ Network gateway_default Created
  ✓ Container mqtt-broker Started
```

[テスト] MQTT brokerの動作確認

```
codespace →/workspaces/prototype-system-masuyayout/gateway (main) $ mosquitto_sub -u masuyayouta -P masuya038 -h localhost -t "#" -v test/d1/ 1
```

```
codespace →/workspaces/prototype-system-masuyayout/gateway (main) $ mosquitto pub -u masuyayouta -P masuya038 -h localhost -t test/d1/ -m "1"
```

MQTT brokerの停止

```
codespace →/workspaces/prototype-system-masuyayout/gateway (main) $ docker compose down
[+] Running 2/2
✓ Container mqtt-broker Removed
✓ Network gateway_default Removed
```

[確認] MQTT brokerのログを見る

```
Usage: docker compose [OPTIONS] COMMAND

Define and run multi-container applications with Docker

Options:
  --all-resources      Include all resources, even those not used by services
  --ansi string        Control when to print ANSI control characters ("never"|"always"|"auto") (default "auto")
  --compatibility       Run compose in backward compatibility mode
  --dry-run            Execute command in dry run mode
  --env-file stringArray Specify an alternate environment file
  -f, --file stringArray Compose configuration files
  --parallel int       Control max parallelism, -1 for unlimited (default -1)
  --profile stringArray Specify a profile to enable
  --progress string     Set type of progress output (auto, tty, plain, json, quiet) (default "auto")
  --project-directory string Specify an alternate working directory
                        (default: the path of the, first specified, Compose file)
  -p, --project-name string Project name

Commands:
  attach      Attach local standard input, output, and error streams to a service's running container
  build       Build or rebuild services
  commit      Create a new image from a service container's changes
  config      Parse, resolve and render compose file in canonical format
  cp          Copy files/folders between a service container and the local filesystem
  create      Creates containers for a service
  down        Stop and remove containers, networks
  events      Receive real time events from containers
  exec        Execute a command in a running container
  export      Export a service container's filesystem as a tar archive
  images      List images used by the created containers
  kill        Force stop service containers
  logs        View output from containers
  ls          List running compose projects
  pause       Pause services
  port        Print the public port for a port binding
  ps          List containers
  publish     Publish compose application
  pull        Pull service images
  push        Push service images
  restart     Restart service containers
  rm          Removes stopped service containers
  run         Run a one-off command on a service
  scale       Scale services
  start       Start services
  stats       Display a live stream of container(s) resource usage statistics
  stop        Stop services
  top         Display the running processes
  unpause     Unpause services
  up          Create and start containers
  version     Show the Docker Compose version information
  wait        Block until containers of all (or specified) services stop.
  watch       Watch build context for service and rebuild/refresh containers when files are updated

Run 'docker compose COMMAND --help' for more information on a command.
unknown docker command: "compose log"
```

ステップ3 エッジデバイスからMQTTブローカーへデータを送る

目標

- ATOM MatrixでMQTTプロトコルによるデータの送信処理を実装する

[設計]テスト項目

- MQTT brokerでsubscribeしているとき、ATOM Matrixを動かすとデータが受信される

[実装] ATOM MatrixでMQTT publishするコードを追加する

[テスト] brokerでデータが受信できるか確認

```

---- シリアル ポート COM3 を開きました ----
ets Jun  8 2016 00:22:57

rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 188777542, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1184
load:0x40078000,len:13260
load:0x40080400,len:3028
entry 0x400805e4
M5Atom initializing...OK
19
E8:9F:6D:08:97:D8
WiFi connected
IP address:
172.16.98.206
Success
Connecting to MQTT...
Publishing at QoS 0, packetId: 0
{"accX":-0.006592,"accY":-0.007568,"accZ":-1.001221}
{"accX":-0.006592,"accY":-0.007568,"accZ":-1.001221}
Connected to MQTT broker: 172.16.98.143, port: 1883
PubTopic: iot_system/y2025/r04i36
Session present: 0
Publishing at QoS 0, packetId: 1
{"accX":-0.008789,"accY":-0.005615,"accZ":-0.99707}
{"accX":-0.008789,"accY":-0.005615,"accZ":-0.99707}
Publishing at QoS 0, packetId: 1
{"accX":-0.008301,"accY":-0.00708,"accZ":-1.000732}
{"accX":-0.008301,"accY":-0.00708,"accZ":-1.000732}

```

```

codespace → /workspaces/prototype-system-masuyayout/gateway (main) $ mosquitto_sub -u masuyayouta -P masuya038 -h localhost -t "#" -v
iot_system/y2025/r04i36 {"accX":-0.008301,"accY":-0.006836,"accZ":-1.000244}
iot_system/y2025/r04i36 {"accX":-0.008789,"accY":-0.005615,"accZ":-0.99707}
iot_system/y2025/r04i36 {"accX":-0.008301,"accY":-0.00708,"accZ":-1.000732}
iot_system/y2025/r04i36 {"accX":-0.005615,"accY":-0.004395,"accZ":-0.998291}
iot_system/y2025/r04i36 {"accX":-0.001709,"accY":-0.005615,"accZ":-0.992188}
iot_system/y2025/r04i36 {"accX":-0.006348,"accY":-0.010254,"accZ":-1.004395}
iot_system/y2025/r04i36 {"accX":-0.001953,"accY":-0.003418,"accZ":-1.001221}

```

ステップ4 AWS IoT Coreの準備

目標

- AWS IoT Coreの設定を行う
 - モノを登録する

- 証明書によるデバイス管理
- ポリシーを理解する

[設定] AWS Academy Learner Lab

1. 「モジュール」をクリックする
2. 「AWS Academy Learner Labを起動する」をクリックする
3. 「Start Lab」をクリックする
4. 左の方にある「AWS」のところが緑の丸になってることを確認し、クリックする(緑ならLearner Labが起動している。黄は起動中。赤は停止。)
5. AWS Consoleにアクセスできる

[Option] AWS CLI/Terraform

AWS CLIやTerraform等で扱う場合には「AWS Details」をクリックし、credentials情報を入手し、利用する

[設定] AWS IoT Coreでデバイス登録(bridgeを登録、証明書、秘密鍵等をダウンロードする)

1. AWS IoT Coreを選択
2. 「管理」「すべてのデバイス」「モノ」を選択
3. 「モノを作成」を選択
4. 「1つのモノを作成」を選択
5. 「モノの名前」を入力([iot_2025_Masuya](#)にしました)
6. 「新しい証明書を自動作成（推奨）」を選択
7. 「ポリシーを作成」をクリック（新しいタブが開いてポリシーを設定する画面に写る）
8. 「ポリシー名」を入力（[iot_2025_Masuya_policy](#)にしました）
9. 「JSON」をクリックし以下のJSONを入力する。

モノの詳細

名前 iot_2025_Masuya	タイプ -
ARN  arn:aws:iot:us-east-1:890916948561:thing/iot_2025_Masuya	請求グループ -

< 属性 証明書 モノのグループ コマンド履歴 Device Shadow アクティビティ パッケージ >

証明書 (1) 情報

このモノのリソースにアタッチされたデバイス証明書。

< 1 > 

☐ 証明書 ID

▲ | ステータス ▼ | 証明書とモノの関連付け ▼

☐ [7ca50ba2c593c688617c77126eb3049b7a9b6f3353de75669af938e59afbb0d2](#)  アクティブ 非独占的

ステップ5 MQTT ブローカーをMQTTブリッジとして機能させ、エッジデバイスからAWS IoT Coreへデータを送る

目標

- MQTT bridgeの役割について理解し、ATOM MatrixからのデータをAWS IoT Coreに送る
 - 証明書によるデバイス管理
 - bridgeするトピックの設定

[設計]テスト項目

1. Gateway (MQTT broker)からMQTTでデータをpublishし、AWS IoT Coreでsubscribeできるか確認する
2. ATOM Matrixからデータをpublishし、AWS IoT Coreでsubscribeできるか確認する

[設定] mosquittoのbrokerをAWS IoT Coreへのbridgeとして設定する

- IoT Coreへのbridgeとして設定するための設定ファイルmqtt-broker/bridge.confを作成し、以下の場所にファイルを配置する

[テスト1] MQTTブローカーとAWS IoT Coreの疎通確認

トピックをサブスクライブする

トピックに公開する

トピックのフィルター | 情報

トピックフィルタは、サブスクライブするトピックを記述します。トピックフィルタには、MQTT ワイルドカード文字を含めることができます。

#

▶ 追加設定

サブスクライブ

サブスクリプション

#

一時停止

クリア

エクスポート

編集

#

① ワイルドカードトピックにメッセージを発行することはできません。
メッセージを発行する別のトピックを選択してください。

▼ localgateway_to_awsiot

May 12, 2025, 16:35:04 (UTC+0900)

② メッセージを指定された形式で表示できません。

test

[テスト2] ATOM Matrixからデータを送る

トピックをサブスクライブする

トピックに公開する

トピックのフィルター | 情報

トピックフィルタは、サブスクライブするトピックを記述します。トピックフィルタには、MQTT ワイルドカード文字を含めることができます。

#

▶ 追加設定

サブスクライブ

サブスクリプション

#

一時停止

クリア

エクスポート

編集

#

① ワイルドカードトピックにメッセージを発行することはできません。
メッセージを発行する別のトピックを選択してください。

▼ iot_system/y2025/r04i36

May 12, 2025, 16:49:43 (UTC+0900)

```
{
  "accX": -0.003174,
  "accY": -0.046143,
  "accZ": -0.995117
}
```

▶ プロパティ

<div><div>▼</div><div>iot_system/y2025/r04i36</div><div>May 12, 2025, 16:50:13 (UTC+0900)</div></div> <div><div>{</div><div>"accX": -0.005615,</div><div>"accY": -0.04834,</div><div>"accZ": -0.997559</div><div>}</div></div> <div><div>▶</div><div>プロパティ</div></div>
<div><div>▼</div><div>iot_system/y2025/r04i36</div><div>May 12, 2025, 16:50:03 (UTC+0900)</div></div> <div><div>{</div><div>"accX": -0.002686,</div><div>"accY": -0.048584,</div><div>"accZ": -0.999512</div><div>}</div></div> <div><div>▶</div><div>プロパティ</div></div>
<div><div>▼</div><div>iot_system/y2025/r04i36</div><div>May 12, 2025, 16:49:53 (UTC+0900)</div></div> <div><div>{</div><div>"accX": -0.004883,</div><div>"accY": -0.048584,</div><div>"accZ": -0.998047</div><div>}</div></div> <div><div>▶</div><div>プロパティ</div></div>

ステップ6 AWS IoT Coreで受信したデータをNodeREDで可視化する

目標

NodeREDで可視化する

- EC2のインスタンスにNodeREDをセットアップする
- NodeREDにダッシュボードノードを追加する
- NodeREDでコードを作成し、可視化する

[設計] テスト項目

- ATOM MatrixからのデータがNodeREDで可視化される

[設定] EC2インスタンスにNode-REDをインストールする

<input checked="" type="checkbox"/>	Name	▼	インスタンス ID	▼	インスタンス...	▼	インスタンス...	▼	ステータスチェック	▼	アラームの状態	▼	アベイラビリティ...	▼	パブリック IPv4 DNS	▼	パブリック I...	▼	Elastic IP
<input checked="" type="checkbox"/>	nodered		i-0a804418923ae8b4e		実行中		t2.micro		初期化しています		アラームを表示		us-east-1c		ec2-98-82-16-111.com...		98.82.16.111		-


[設定] AWS IoT CoreにNodeREDのためのデバイスを追加登録する

モノの詳細

名前

node-red

ARN

arn:aws:iot:us-east-1:890916948561:thing/node-red

タイプ

-

請求グループ

-

<

属性

証明書

モノのグループ

コマンド履歴


Device Shadow

アクティビティ

パッケージ

>

証明書 (1) 情報



アクション ▼


デタッチ

証明書の作成

このモノのリソースにアタッチされたデバイス証明書。

Q 証明書を検索

< 1 >




☐ 証明書 ID

▲

ステータス ▼

証明書とモノの関連付け ▼

☐ [e67d74edeb6d80e3024038b4c2b16e7ad4b3af1a28e937efde09ba5b5c86342f](#)

 アクティブ

非独占的

☐ 名前

モノのタイプ


☐ [node-red](#)

-

☐ [iot_2025_Masuya](#)

-

[実装] AWS IoT CoreからNodeREDにデータを送る(NodeREDでsubscribeする)

 #

接続済

debug 1

2025/5/19 15:41:47 ノード: debug 1

test/test : msg.payload : Object

▶ { accX: 1, accY: 2, accZ: 3 }

2025/5/19 15:43:26 ノード: debug 1

test/test : msg.payload : Object

▶ { accX: 1, accY: 2, accZ: 3 }

2025/5/19 15:45:58 ノード: debug 1

test/test : msg.payload : Object

▶ { accX: 1, accY: 2, accZ: 3 }

2025/5/19 15:45:58 ノード: debug 1

test/test : msg.payload : Object

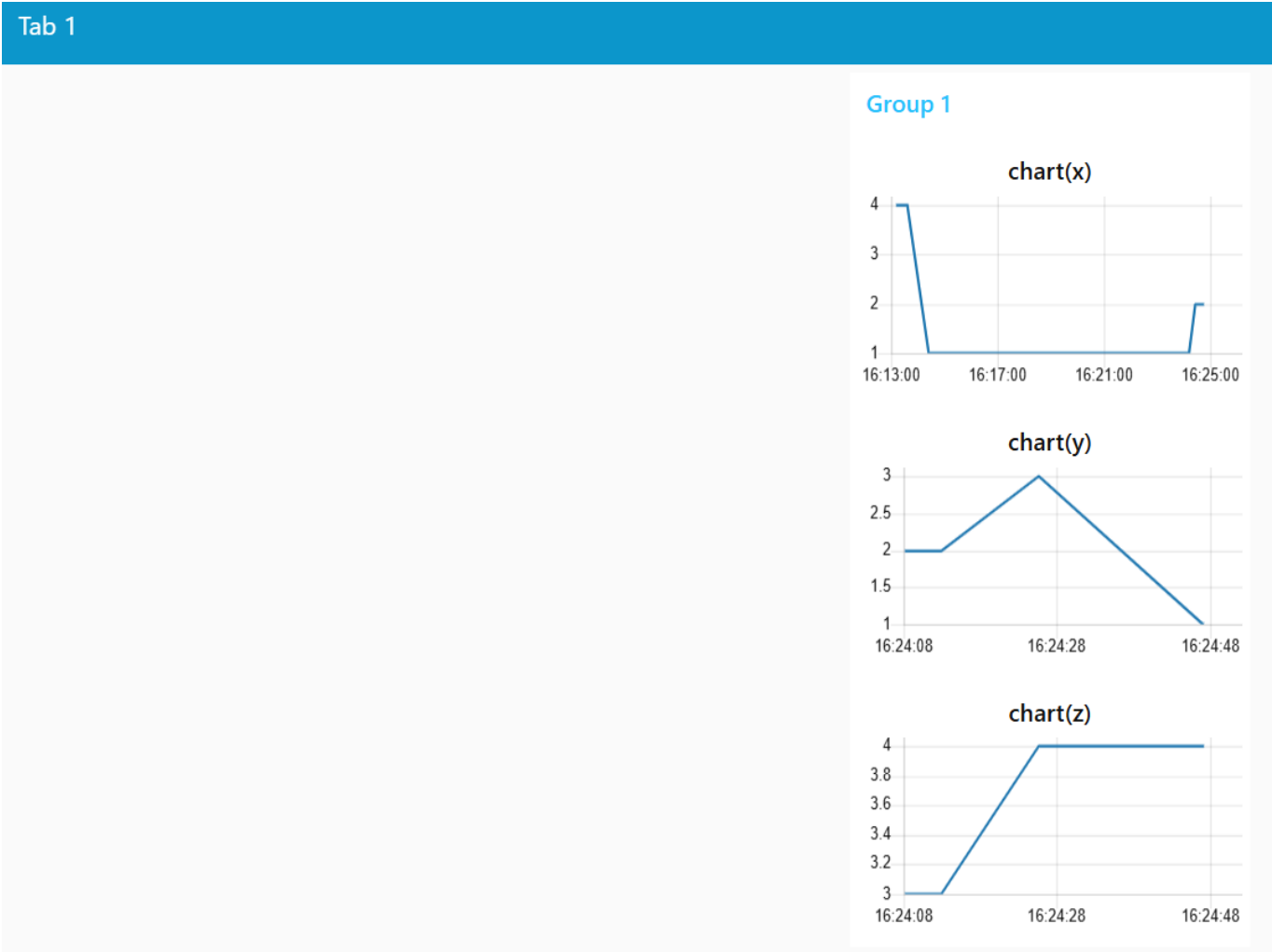
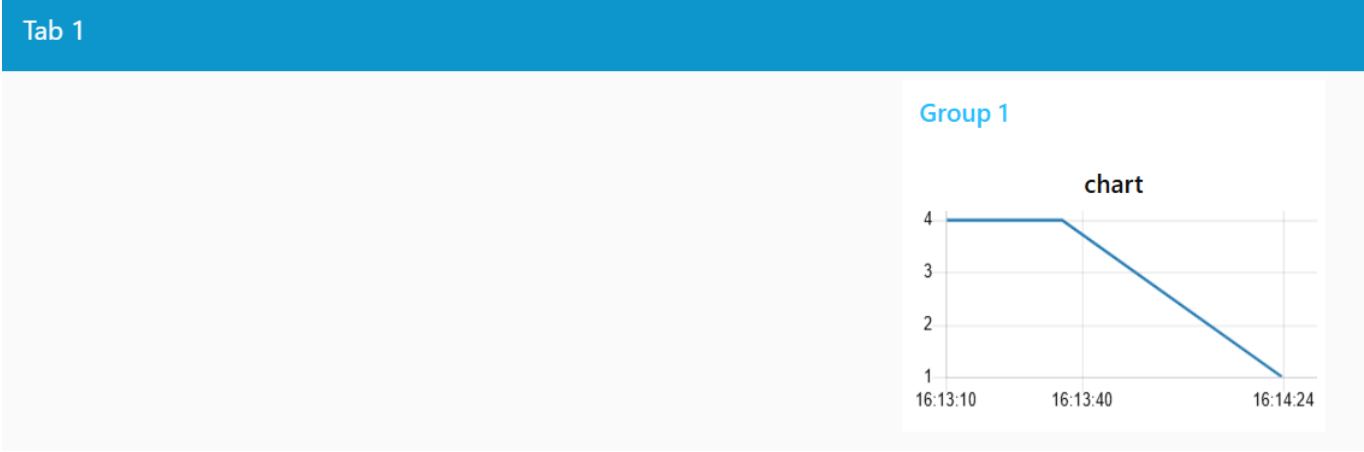
▶ { accX: 1, accY: 2, accZ: 3 }

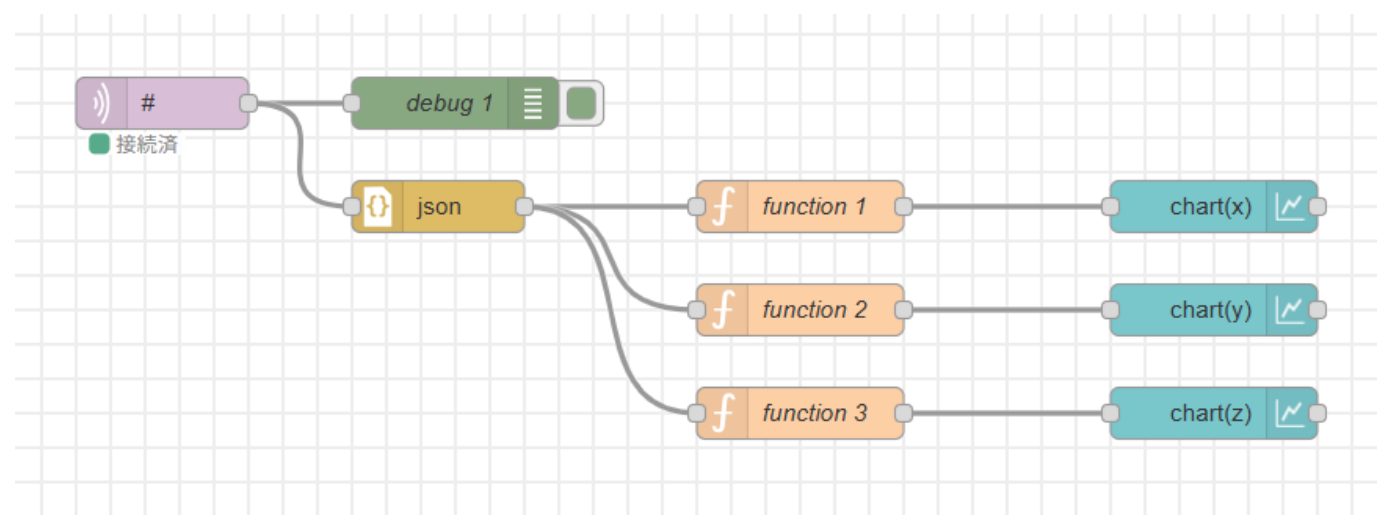
2025/5/19 15:46:01 ノード: debug 1

test/test : msg.payload : Object

▶ { accX: 1, accY: 2, accZ: 3 }

[実装] NodeREDで可視化する





[テスト] Atom Matrix～MQTT bridge～AWS IoT Core～Node-REDとデータが送信され、可視化できることを確認する

