

2 モノのインターネット (Internet of Things)

目次

1. 実験目的および注意事項.....	1
2. 背景と実験理論	3
3. 言語仕様	14
4. 実験準備	20
5. 設計と実装.....	23
6. レポートとディスカッション	36
7. 実験環境の初期インストール	39

1. 実験目的および注意事項

1.1. 実験目的

本実験ではモノのインターネットを実際に構築し、そのアプリケーションについて考える。まず、モノのインターネットは、Internet of Things の和訳であるが、その略語である IoT という呼び名の方がよく利用されるため、以降 IoT と呼称する。

IoT を理解するため、低消費電力の小型ノードを用いること、そのノードに搭載された各種センサを用いて物理世界の情報を取得・集約し、インターネットを通してクラウドなどにおいて提供されるサービスアプリケーションに提供するといった一連の仕組みを実際に構築し、またその処理について理解する。また、IoT においてよく利用されている簡便な無線通信網も利用し、その動作内容について理解する。これらを通して、IoT を理解し、自ら IoT システムを構築できる知識と能力、さらにその運用能力を身に着ける。

1.2. 実験の位置づけ

システムデザイン工学科では、システムを解析・設計合成・調和評価という 3 つの軸で捉え考えている。その根幹は、システムの表現であり物理世界の数値化である。IoT は物理世界からの様々な情報取得し(観察)、仮想世界において処理・知覚化・最適化・予測などを行い(状況判断と意思決定)、再び制御やアクチュエーションを通して物理世界に作用し(実行)、さらにその結果が情報取得される。この Observe(観察)・Orient(状況判断)・Decide(意思決定)・Act(実行)のループは OODA ループと呼ばれシステムの解析・設計合成・調和評価すべてに通じる手法である。本実験では、このループの構築における基本となる IoT システムを実装し動作を確認する。

具体的には、センサをコントローラに接続する回路設計、コントローラのプログラムを記述するプログラミング設計、センサで取得したデータを集約する Zigbee を用いた無線通信網設計、その結果をパソコン上でデータの視覚化など様々なサービスを構築するアプリケーションプログラム設計といった一連の流れ全体を記述し確認する。さらに言えば、センサやコントローラといったインターネットの末端やエッジ領域のデバイスから、それらを利用するクラウドサービスまで、インターネットサービス全体を掌握する IoT システム設計をハンズオンで学ぶ実験である。

このうち、プログラミングについては、基本的なプログラムの記述スタイルや動作について 2 年生の

プログラミング演習の授業で学んだ。無線通信網設計については、マルチメディアデザインの授業で詳しく説明するセンサネットワークへ引き継がれる。データを扱うアプリケーションプログラム設計は、データシステムの知能化とデザインの授業でさらに発展的内容を学ぶ。システム全体のアーキテクチャ設計は情報処理システムに、複数のコントローラや IoT システムの協調動作は分散処理システムへと引き継がれる。本実験は、これらの授業とも密接に関連しており、これらの授業間の連携を実体験をもって認知、確認することを目指している。

これらのうち、特にプログラミングに関して、システムデザイン工学科では、2 年生の必須授業であるプログラミング演習で学んだ Python など、スクリプト言語のみ扱っており、コンパイラ言語は学んでいない。本実験が取り上げる IoT システムの開発では、CPU 能力が低く、低消費電力であることが求められ、メモリ搭載量やストレージ搭載量が限られる。このように、リソース制限が厳しい組み込みデバイスの設計では、リソース管理も含めた特別なコンパイラ言語を用いたプログラム設計が求められ、一般的なスクリプト言語とは異なる設計知識・スタイルが必要となる。なお、これは、大型計算機においても、処理速度最適化や負荷最適化を図る場合、ハードウェアに密接にかかわる部位の設計を行う場合に必要なスキルである。本実験では、IoT システムにおけるセンサ・アクチュエータノード(トランスデューサノード)として Arduino を利用する。また、その開発にはコンパイラ言語である C/C++をベースとした独自仕様言語を用いる。リソース管理も含めたプログラミングが求められるが、本実験では基本的な内容に留めており、既習のスクリプト言語設計の知識で十分対応できる。より高い記述能力や記述自由度、リソース管理が必要となる無線モジュールの操作や、各種専用デジタルデバイスの制御などについては専用ライブラリを提供し本実験では直接触れない。しかしながら、すべてバイナリではなく、コードを提供するため、各自で内容を確認することができる。その動作内容を理解するとは、デバイスドライバ設計の基本を学ぶことにつながるのでぜひチャレンジしていただきたい。

1.3. 注意事項

本実験はレポートなど本テーマに関する連絡手段として Slack を利用する。Slack では、ワークグループとチャネルという階層があり、実験を受ける際にまず本実験のワークグループに参加すること。本実験のワークグループでは次の「質問箱」と「サポート」の 2 チャネルを主に利用する。Slack を利用するために必要なアカウント作成やログインなどは、同時に配布されるパワーポイント資料を参照すること。

- 質問箱

本実験テーマに関する質問を受け付ける。また、質問はこのワークスペースにアクセスしている全学生に公開される。また、匿名とならない点に注意する。質問することは重要であり、その回答を通して、同様に疑問に思っている学生の学習が進む。質問など利用形態は採点に影響しない。初步的な質問・疑問でも、質問した学生の行動を称える。

- サポート

本実験テーマに関する連絡や参考事項など、様々な追加情報が掲載される。レポートの内容も掲載される。特に重要な内容は LMS でも掲載するが、基本的には Slack で情報を公開するため、必ず確認すること。

2. 背景と実験理論

本実験で取り上げる IoT について概要を述べる。

2.1. IoT とは何か

IoT つまり「モノのインターネット」は、その名前の通り様々な「モノ（物）」がインターネットに接続され、インターネットを介して何かしらのサービスアプリケーションと情報を交換する仕組みや技術を意味する。IoT は高度情報化社会を形作る上での基本構成要素であり、ソサエティー5.0、インダストリー4.0、スマートシティ、デジタルトランスフォーメーション、デジタルツインなどの様々なバズワードで表現される各種新技術や新サービスにおける基盤技術の一つである。IoT という言葉は、1999 年に MIT の AutoID ラボ創始者のひとりケビン・アシュトン氏により、「ユビキタスセンサを通してインターネットが物理世界を繋ぐシステム」の名称として提示された用語を起源としている。なによりも、IoT は実世界に近い場所で情報を扱うことができるため、最も新鮮な情報を取得・利用できるデバイスでもある。この地の利こそが IoT の本質である。

2.2. なぜ IoT なのか

IoT という用語には、バズワードとして様々な意味が込められており、その全容を正確に理解することは困難であるが、その背景を知ることで、より的確に IoT という用語を理解できるであろう。まず、IoT を純粋に実現しようとすれば、「モノ」つまり物理世界を対象とし、物理世界の情報を取得する温度計といったセンサや、物理世界に作用するモータといったアクチュエータが存在する環境を想定する必要がある。IoT ではこれらをインターネットに接続するための通信手段を備え、加えて何かしらの計算資源も備わっていることが必要である。

しかしながら、同様の構成要素を持つシステムという観点では、センサネットワークや、ユビキタスコンピューティングも該当するため、センサやアクチュエータ、インターネット接続、計算資源を備えるといった説明では IoT とこれらを区別することができない。センサネットワークはセンサやセンサ群をネットワークで結び、その中心となるノードやインターネットへ情報を集約し提供するシステムである。ユビキタスコンピューティングは、我々の生活環境に組み込み計算ノードが存在を主張せず至るところに多数埋め込まれた環境を構築し、各種計算やセンシングを行うシステムである。では、IoT とは何が違うのであろうか。

センサネットワークやユビキタスコンピューティングといった用語は基本的にシステム構築論に主眼を置いた用語である。すなわち、センサネットワークであればその構築に必要なセンサ技術や通信技術、設置や電源確保といった技術課題と解決が議論される。ユビキタスコンピューティングであれば不可視性の確保と同様に通信環境や電源確保、さらには分散協調制御といった運用面の技術的課題と解決が議論される。IoT ではこのようなセンサやネットワークに関連する技術的側面よりもサービス提供、つまり、IoT システムを同様に利用してどのようなメリットを得るのかという議論に重きが置かれる。このサービス化の流れはセンサネットワークから IoT への変化に限ったことではない。電力システムからスマートグリッド、クラスタコンピューティングからクラウドなど、枚挙に暇がない。

IoT が社会必須インフラとなった今は、IoT がどのようなものかを知らずとも、また IoT の構築技術を気にしなくとも、IoT を利用したサービスを受けることができる。この可用性を手に入れるには、物理層、ネットワーク層、そしてアプリケーション層、さらにはオペレーションやサービスなど、すべてのレベルで技術革新が必要であった。電子回路や電気回路の製造技術に進歩がなければ、小型化・低消

費電力化が達成できない。無線通信の進歩がなければ、設置利便性が達成できない。リチウムイオンなどバッテリ性能の向上がなければ、すぐに利用できなくなるであろうし、オペレーティングシステムや開発環境の進歩がなければ、取扱いが面倒なままであったであろう。分散協調技術がなければ、応用範囲が乏しく、魅力のない技術で終わっていたであろう。さらに、低価格化することで普及が一層進むとともに、様々な利用例が蓄積されることで必須インフラとして認知されるに至った。現在も技術革新は絶え間なく続いている、性能や可用性の向上が進められている。具体的な可用性については第 2.5.1 節センサネットワークで述べる。

2.3. IoT サービス

マルチメディアデザインの授業でも扱うが、次の事例に示すように、情報通信技術により高機能化・高効率化を実現するスマートインフラにおいて IoT が利用されている。

- スマートシティインフラ：都市・建物のサービスや住民向けサービスとしてスマートコミュニティ/スマートタウン/スマートビルディング/ハスマートウスなど
- エネルギインフラ：HEMS/BEMS などエネルギー管理システムによる制御・空調など HVAC 制御・系統電力制御など
- 交通インフラ：GPS・自動運転・交通制御など
- セキュリティインフラ：犯罪防止のためのライト/カメラ/警報/施錠など
- 災害対応インフラ：モニタリングや避難指示など
- 医療健康インフラ：遠隔診察や遠隔診療、生活モニタリングなど
- 農業インフラ：圃場モニタリング/営農リコメンデーション/農薬散布など

これら典型的なスマートシティ・スマートハウスにおける IoT の事例として、矢上キャンパスにも近い綱島スマートサステナブルタウンの事例を紹介する。次の URL にアクセスすると、街の様々な情報が提供され、利用されていることがわかるであろう(<https://tsst.scim-service.jp>)。ハードウェア設計やソフトウェア設計だけでなく、どのように利用されるか、何がもたらせるのかが問われるため、極めてシステムデザイン工学的な内容といえる。

2.4. IoT のコントローラ

一般に IoT を構築するには、何かしらの計算を行い制御するデバイス、すなわちマイクロコントローラが必要である。マイクロコントローラは電源電圧や信号入出力電圧、供給可能電力などに対する制約が厳しいため、直接センサなど外部のデバイスや素子と接続することが困難であり、レベルを合わせ、保護するための抵抗やトランジスタなど様々な付属回路も実装する必要があった。また、接続できたとしても、実際にそれらデバイスや素子を扱うためには、マイクロコントローラであるがゆえにオペレーティングシステムなどは搭載されておらず、本来 OS が担う入出力やメモリ管理を含むすべての基本機能を記述しなければならない。OS がなければプロセスの概念もないためすべての機能を一つのプログラムとして実装しなければならない。低消費電力化には効率の良いアルゴリズムが必要であり、細やかなリソース管理も OS のサポート無しにすべて記述しなければならない。さらに、その記述にはマシン語と呼ばれるプリミティブな言語や C 言語による設計が必要であり、簡単に試すことや学ぶことができる状態ではなかった。

まず、マイクロコントローラ側の発展として、Microchip 社の PIC と呼ばれるデバイスが登場する。PIC は入出力を強化、汎用化、入力出力相互入替え可能とすることで、センサなど外部デバイスの接続

性を高めた。PIC の成功を受けてさらに扱いやすい Atmel 社による AVR が登場する。AVR は開発環境も無料で提供されたため、普及が一層広まった。AVR により入受けの接続性はかなり高まったが、利用には、電源回路やクロック発生回路、リセットスイッチ、など様々な周辺回路やデバイスが必要であり、これらと一緒に実装する必要があった。また、その設計も C 言語が主に利用されていた。これら、PIC や AVR といったマイクロコントローラは、価格にして 100 円強であり、身の回りの様々な家電などで大量に利用されている。

このような中、これらの周辺回路を混載したワンボードデバイスで、より平易な開発環境も無償提供する製品が登場するようになった。例えば、Parallax 社による BASIC Stamp は PIC を搭載し、フリーで提供される開発環境は BASIC と呼ばれる初心者にも扱いやすい言語を利用して設計できる。現在最も広く普及しているのが AVR を搭載した Arduino である。Arduino はその設計環境のみならず、基板設計図もフリーであり、様々な互換品が提供されている。その他、Arduino と同じ設計環境を利用でき、無線デバイスも混載した Espressif Systems 社の NodeMCU など、Arduino エコシステムはさらに広がりつつある。

Arduino はワンボードデバイスで半田付けが不要、かつ構造が極めてシンプルな組み込みマイクロコントローラボードである。C/C++言語を用いた設計を基本としているが、ライブラリが充実し、比較的簡単にセンサやアクチュエータを用いた設計ができるように工夫されている。プログラミングに興味さえあれば、年齢を問わず IoT システムを構築できる手軽さを備えている。Arduino に搭載されている比較的高性能な AVR マイコンは 500 円程度で Arduino は互換品であればおよそ 3,000 円弱で購入できる。

Arduino の開発には Arduino IDE と呼ばれる設計環境を PC にインストールし、Arduino と PC を USB で接続して行う。IDE とは Integrated Development Environment、すなわち統合開発環境のことと、プログラムのソースコードを記述するためのソースコードエディタ機能、コンパイラやリンカを呼び出して実行可能ファイルを構築するコンパイル・ビルト機能、エラー箇所のハイライトといったテストやデバッグ支援機能、関連ライブラリの管理機能、その他プログラム用例集(サンプルプログラム)やマニュアルなどが統合されている。Arduino IDE にはシミュレーションという概念はなく、実機を使って実際に動かして試す必要がある。本来、いきなり実機を用いて設計したプログラムを試すのはエラーがデバイスの破壊に直結するため、通常は採用しない方法である。しかしながら、Arduino は、マイクロコントローラが堅牢であることから、このような構成でも十分実用に耐えることができる。なお、デザイン・アート・ファッショ・建築など多様な分野で利用され、マルチメディアデザインの授業でも扱う Processing と呼ばれるグラフィクスツール IDE は Arduino IDE と類似するインターフェースを持つが、開発言語は Java である。汎用 IDE では Eclipse が著名である。

2.5. IoT における無線通信と ZigBee

2.5.1. センサネットワーク

センサネットワークは、その利用形態から次の特徴を有する必要がある。

- どこでも設置

物理世界と直に接触する。したがって、埃・風雨・氷雪等にさらされる場所でも設置できなければならず、密閉技術、温度保障技術などパッケージング技術が必要となる。特に電源コンセントや通信環境がない場所での設置を想定する場合、電力消費や確保に関する制約は技術的に厳しく、長時間バッテリ駆動を可能とする低消費電力、環境エネルギー利用による自己発電技術(エナジーハーベスティング)が

必要となる。さらに、設置性を高めるのであれば、有線よりも無線通信の利用が望ましいであろう。しかしながら、電波は経路上にある物体の影響を受けやすく、後述するアドホックネットワーク技術や高信頼通信技術が必要となる。

- 誰でも利用

基本操作に関して、情報処理に接していない誰でもが使えるようにしなければならない。ネットワークアドレスやネットワーク経路などネットワークに関わる一切の設定を不要にすることが理想であるが、セキュリティの観点もあるため、ある程度の設定を行えば、比較的自由に設置できることが必要である。例えば、センサノードを追加する際に、他のノードすべてをプログラムしなおすといった事態は避けなければならない。その他、ノードの動作状況管理やノードの故障検出と通知、さらには故障しても全体として継続動作させる技術も必要である。

どこでも設置、誰でも利用、すなわち可用性の向上が求められているといえる。

2.5.2. アドホックネットワーク

アドホックネットワークは、ノードの移動を許容し、ノードの追加や消滅も許容する技術として提案・開発された。その他、通信外乱による影響の隠ぺいや、環境により通信状況が変化する場合など、システムの安定化にも寄与する。

アドホックネットワークの本質は、経路制御プロトコルを用いた経路最適化である。一般に通信経路が複数ある場合でも、実際に用いる経路は一つに固定される。しかしながら、何かしらのトラブルがあった場合や、ノードが移動した場合は、それまで通信の中継を行っていたノードと通信できなくなる。そのような場合でも、経路制御プロトコルにより適切に新たな中継ノードと通信し、全体として情報伝達を滞りなく行うことができる仕組みが必要となる。これを達成するのがアドホックネットワークである。

経路を適切に保つため、通信を始める前に経路を決定するリアクティブ型プロトコルと、定期的に経路に関する情報交換を行うことで都度経路を決定するプロアクティブ型プロトコルが存在する。リアクティブとプロアクティブの本質的な違いは、ルーティングプロトコルをいつ交換するかであり、経路決定手順に違いはない。

2.5.3. センサネットワークの発展

IoT 発展の礎となったセンサネットワークに関する研究は古くから行われていたが、1999 年米国 DARPA によるマイクロ電子機械システム(MEMS: Micro Electro Mechanical System)によるスマートダストプロジェクトにより大きく発展した。スマートダストプロジェクトは、演算処理、センシング、通信、電源を混載した体積 $1.5mm^3$ 以下、重さ 5mg 以下の超小型センサネットワークデバイスを MEMS で構成することを目指した挑戦的なプロジェクトであった。結局、当初の目標が技術的に厳しく、要求仕様を満たすデバイスは提案されなかった。一方で、軍需を含め様々な応用が提案されただけでなく、ユビキタスコンピューティングといった新たな計算資源の概念を生み出し、デバイス技術(MEMS)、無線・有線ネットワーク構築技術など、様々な領域に活発な議論をもたらした。その後、現実的な観点から COTS Dust と呼ばれるプラットフォーム開発が行われ、DARPA の NEST(Network Embedded Software Technology)プロジェクトにより MICA MOTE として一般販売されるに至った。MOTE は当時としては画期的に小さく、単 3 電池 2 本とほぼ同じサイズで約 1 年動作可能な無線センサノードであり、アドホックネットワークの構築が可能であった。MOTE はセンサネットワークプラットフォームの

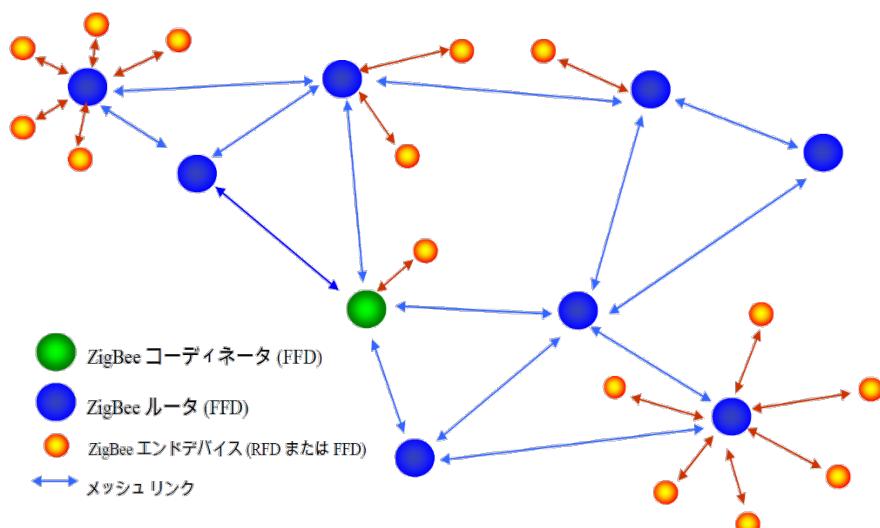
入手を容易にし、様々な領域で革新的な応用がなされた。これらの技術の上に ZigBee が提案された。その後通信距離を飛躍的に拡大した LPWA といった新たな技術も提案されている。

2.5.4. ZigBee の特徴

ZigBee は次のようなセンサネットワーク用途に適した特長を備えている。

- メッシュ型のネットワーク構造をサポートし、アドホックネットワークを構築可能である。
- メッシュ型のネットワーク構造を利用できるため、電波が届かない場所にあるセンサノードを追加する際にも、中継ノードとしてルータデバイスを適宜配置できる。つまり、通信エリアを容易に拡大できる。
- 複数の経路を管理しているため、障害発生時も直ちに別の経路に変更できる。したがって障害に強いネットワークを構築できる。
- 低消費電力な無線通信を用いており、エンドデバイスの省電力化が可能である。
- ZigBee は単なる物理層の規格でなく、アプリケーション層も規格化されている。どのようなエンドデバイスでも設置すれば動作するという手軽さがあり、特に海外において広く普及している。これは、Bluetooth も同様であり、例えば Bluetooth 対応の無線ヘッドセットであればメーカーが異なっても接続可能である。

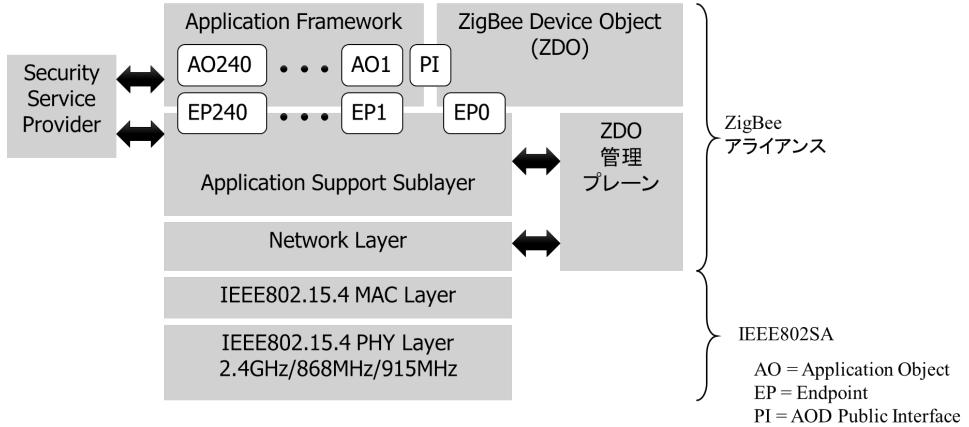
エンドデバイスとは、センサを保有し、特に低消費電力化が求められるデバイスである。基本的にはスリープ状態にあり、計測時にスリープ状態から抜けてセンシングとデータの送信を行い、また直ちにスリープ状態に移行するといった低消費電力動作を行う。このようなエンドデバイスは、電池による電源供給下でも長時間動作することができる。ZigBee は下図の通りエンドデバイスの他に中継ノードであるルータノード、およびデータを集約するコーディネータと呼ばれるノードが存在する。ルータやコーディネータもセンサを保有できるが、ルータは中継機能を追加で備えており、コーディネータは通常インターネットなど外部ネットワークとの接続手段をさらに追加で備えている。本実験は ZigBee を用いた無線通信網を利用し、センサデータを、コーディネータを介して集約、パソコンに取り込んで、センサデータを利用したアプリケーションを設計する。



2.5.5. ZigBee のアーキテクチャ

ZigBee は ZigBee アライアンスによるアプリケーション層からネットワーク層までを規定する

ZigBee アライアンスと、データリンク層および物理層を規定する IEEE802.15.4 の 2 つの標準に基づいて構築されている。なお、特に区別せず、まとめて ZigBee と呼ぶことが多い。次の図のような構造を有し、センサなど複数の ZigBee Device Object は、相互に接続する Application Support Sublayer を介して Network Layer により ZigBee プロトコルとして無線相互通信が行なわれる。



既に述べた通り、ZigBee には、コーディネータ、ルータ、エンドデバイスといったデバイスタイプ (ZigBee Logical Device Type)がある。他にもアプリケーションから見たデバイスタイプとして、センサやアクチュエータの種類、例えば、照度センサや、温度センサ、空調コントローラといった種別 (ZigBee Application Device Type)がある。さらに、Zigbee Logical Device Type のうち、どの機能を実現できるかによっても区別でき、すべての機能を搭載できる、つまりコーディネータにもルータにもエンドデバイスにもなれる Full Function Device と、いずれかの機能が利用できない Reduced Function Device の種別 (ZigBee Physical Device Type)がある。

2.6. IoT システムのソフトウェア設計

本実験では、IoT における必須となる 3 つの設計、つまり、IoT センサノードのソフトウェア設計と、IoT センサネットワークのソフトウェア設計、そして、一般にクラウドで実行される IoT サービスアプリケーションのソフトウェア設計を扱う。その概要について、説明する。

2.6.1. IoT センサノードのソフトウェア設計

IoT センサノードのソフトウェア設計について、後述する Arduino IDE を用いて IoT センサノードつまり Arduino を設計する。設計に用いるプログラミング言語として、C/C++を基本とした Arduino 言語を利用する。Arduino 言語は比較的初心者でも扱いやすい文法構造を有する。その文法については第 3 章言語仕様で説明する。IoT センサノードの設計は、特にメモリサイズが小さいため簡潔かつ実行効率や、メモリ利用効率の高いプログラミングを行うことが求められる。また、一般にリソース制約が厳しく、非力なマイクロコントローラを利用するため、高度な処理、複雑な処理を実現することは避けなければならない。なお、Arduino 言語はコンパイル型言語であるため、スクリプト言語よりも実行速度が速い。ただし、周辺機能もすべて含んだ場合、例えばスクリプト言語では、スクリプト言語のインターフェリタも必要となることから、実際にはかなりのメモリを消費する。コンパイル型言語では、必要なライブラリのみリンクが選択的にバイナリに含めるため、メモリ利用効率が高くなる。このような理由から、IoT センサノードのソフトウェア設計ではコンパイル型を用いることが一般的である。

2.6.2. IoT センサネットワークのソフトウェア設計

ZigBee を実現する XBee 通信モジュールにもマイクロコントローラが存在する。このマイクロコントローラはメーカーが提供する firmware を用いて動作させるため開発対象とはならない。しかしながら、各種パラメータは設定可能であり、専用ツールを用いて ZigBee モジュールのパラメータを変更することで動作内容を変更できる。したがって、本実験では直接 IoT センサネットワークのソフトウェア設計を行わない。パラメータの設定については、第 7.2 節 XBee の設定で説明する。

2.6.3. IoT サービスアプリケーションのソフトウェア設計

本実験では IoT アプリケーション設計に Node-RED を用いる。サービスアプリケーションは一般にクラウドで実装されるが、ここでは、Windows PC 上にサービスアプリケーションとして Node-RED を実装し、サーバとして稼働させることで実現する。Node-RED はクラウドサービスでもよく利用されるサービスアプリケーションであり、web サーバとして動作する。したがって、そのクライアント、すなわち Node-RED を設計したり、設計した結果を見たりする端末は、web ブラウザ機能があれば何でもよく、インターネットに接続さえしていれば、どこからでもアクセス可能となる。この web サーバとして動作するという点は、クラウドサービスを構築する観点からみても重要な特長である。ただし、セキュリティには厳重に注意する必要がある。本実験では、各 PC でサービスを構築し、実験室環境内のみ共有され、インターネット外部から見えないようにしている。

Node-RED は JavaScript を用いて設計されており、Node-RED を用いたアプリケーション設計においても、JavaScript を用いることができるが、これは Node-RED のあるべき利用スタイルではない。Node-RED は GUI を用いたビジュアルプログラミング環境を提供しており、データの流れであるフローをブロックと配線でつなげて設計する、フローベースの開発ツールである。したがって、特に JavaScript に精通していないとも、また JavaScript に触れることなくアプリケーションを設計可能である。特に Node-RED はすべて web ブラウザ上で設計・実行が可能であるという特長を有し、プログラミングに精通していないとも、設計することができる。ランタイムは Node.js 上に構築されており、web サーバとして動作する。ビジュアルプログラミングを行うため、記述の柔軟性や表現力は一般的なテキストプログラミングの形態に劣り、設計の修正コストも大きくなる。

Node-RED で作成されたフロー、すなわち設計図としてのプログラムは JSON 方式で保存また再現できる。この JSON ファイルを理解、また直接編集する必要はなく、また、るべきではない。

このように、本実験では、C/C++、Javascript などの多様な言語を用いるが、IoT では一般的に適材適所で様々なプログラミング言語を用いる。Node-RED の設計については、第 5.8 節 Node-RED によるアプリケーション実装で概要を説明する。

2.7. IoT システムのハードウェア設計

既に年間のセンサ出荷数は 1 兆個を超えており、その種類も多様かつ、精度向上と低価格化が図られている。本実験では Elgoo 社のスターターキットに含まれているデバイスや素子を利用する。オリジナルのキットから実験に利用しない一部デバイスを除外し、その他必要となる USB ハブや無線通信を実現するための ZigBee デバイスである XBee、XBee を Arduino や PC に接続するためのボードなどを追加している。また、構成要素の多いキット 1 組と、少ないキット 2 組を利用し、最大 3 台の IoT 環境を構築する。このように、独自のキット構成であるが、キットに含まれるデバイスや素子、追加したデバイスや素子は、すべてキットとしてではなく、パーツとして一般に入手可能である。

スターターキットの内容物を説明する前に、その扱いにおいて、次の 3 つの点に注意すること。

- デバイスや素子は慎重かつ丁寧に扱い、ピンなど尖っているため、怪我などには十分注意すること。また、すべて再利用するため、破損や曲げ、紛失などに注意すること。
- 鍵括弧[]の中にキット内の個数を示すので、最後に確認し紛失しないようにすること。
- スターターキットのケースへの収納の仕方について後述しているので必ず確認し、元通りきれいに片づけること。
- 3 つのキットケースを配布するが、キットケース間でもともと含まれていないパーツを入れたり、内容量(パーツの個数)が合わないように入れないこと。

以下、センサ、アクチュエータ・インジケータ、抵抗・コンデンサ、電源・配線・その他デバイスにキット内容を説明する。

2.7.1. センサ

本実験では次のセンサを利用可能である。ここにないセンサも、おおよそ同様のインターフェースを持つため、未知のセンサであっても接続の参考となるであろう。

温度・湿度センサ [1] (DHT11)  デジタル通信によりパケットの形で計測値を受け取る。	サーミスタ Thermistor [1]  温度を計測する。アナログセンサであるため、A/D コンバータを介して計測値を受け取る。	光センサ Photo resistor [2]  人間の視覚特性に近い安価なセンサであるが、Cd を利用するため RoHS 指令により利用や販売が制限されている。
プッシュボタン Push Button [5]  内側を向いている 2 組足の対がそれぞれ ON/OFF する。	傾斜センサ Tilt Ball Switch [1]  小さな傾きで内部の鉄球が動くことで ON/OFF する。	ジョイスティック Joystick Module [1]  アナログ値で傾きや押しを読み取る。
ポテンショメータ・ボリューム Potentiometer [1]  あらゆるアナログセンサの基本であり、A/D コンバータや RC 時定数回路を用いて値を読む。	テンキーパッド Membrane Switch Module [1]  4×4 のマトリックススイッチである。	ロータリーエンコーダ Rotary Encoder Module [1]  回転に応じてパルスを生成する。回転角度や速度、加速度などを読み取る。
人感(動き)センサ HC-SR501 PIR Motion Sensor Module [1]  熱源の動きを感じる。	赤外線レシーバ・リモコン IR Receiver · Remote SW [1]  デジタルでボタン情報を読み取ることができる。	RFID モジュール・タグ RFID Module (RC522) [1]  RFID を読み取ることができる。
Sound Sensor Module [1]  アンプ付きマイクと同様の構造で、増幅率を調整できる。	Ultrasonic Sensor [1]  超音波発信器と受信器のセットで、測距が可能である。	水位センサ Water Level Detection Sensor Module [1]  水の導電性を利用して、抵抗値の変化で水位を計測できる。
三軸加速度(ジャイロ)センサ GY-521 Module [1]  X 軸 Y 軸 Z 軸周りの 3 軸の回転加速度をデジタル計測する。積分することで速度や位置も推定できるが精度は落ちる。	リアルタイムクロックモジュール (DS1307) RTC Module [1]  水晶発振器と電池を備え、正確な時刻を刻み続ける時計モジュール。Arduino は時計がないため、時刻管理に用いる。	

2.7.2. アクチュエータ・インジケータ

様々なアクチュエータが存在するが、広義には何かしら環境に働きかけることができるデバイスであり、モータが代表例である。

<p>LED 黄・青・緑・赤・白 [5] 極性があるので注意する。足の長い方が+(アノード)である。</p>	<p>フルカラー(RGB) LED [1] RGB の 3 つの LED が一つに実装されている。アノードコモンタイプである。</p>	<p>LCD モジュール (LCD1602) [1] デジタル信号でキャラクタ(文字)を表示可能である。表示場所など細やかな制御が可能。</p>
<p>7 セグ(セグメント)LED 1 digit 7-segment Display [1]</p> <p>7 個の LED により数字を表現できる。実際には小数点も含めて 8 個存在し、アノードコモンで 2 個と 8 個のカソード、合計 10 個の端子がある。</p>	<p>4 連 7 セグ LED 4 digit 7-segment Display [1]</p> <p>7 セグ LED を 4 つ並べた構造となっているが、どの 7 セグを点灯するかを切り替えるアクティブ点灯方式により単指数は 13 個である。</p>	<p>LED アレイ (MAX7219) [1] シフトレジスタの利用により 8 個の LED 列とシフト信号線、アノード線の 10 本のピンで制御できる。</p>
<p>3~6V DC モータ・ファン 3-6V DC Motor+Fan Blade [1]</p> <p>普通の模型用直流モータである。</p>	<p>ステッピングモータ・ドライバ Stepper Motor / Driver Board (ULN2003) [1]</p> <p>正確に回転角度を制御可能なモータである。駆動には専用のドライバを必要とする。</p>	<p>サーボモータ Servo Motor (SG90) [1]</p> <p>正確に移動確度を制御可能なモータである。回転はできない。角度を PWM 方式で制御する。</p>
<p>アクティブ・パッシブブザー Active / Passive Buzzer [1]</p> <p>アクティブブザーは電圧をかけると決まった音が鳴る。パッシブブザーは電圧を制御して波を作るとその音が鳴る。</p>	<p>メカニカルリレー Relay [1]</p> <p>メカニカルなため反応速度は速くないが AC100V なら 1200W と比較的大電流・大電圧を制御できる。</p>	<p>UNO R3 ボード上の LED [1]</p> <p>13 番 PIN として実装されており、普通の LED として制御できる。</p>

なお、表示装置は一般にアクチュエータではないが、ここでは警告を発するなど人に情報を提供し人の行動変容を生み出すという意味で広義にアクチュエータと定義する。情報を取得するセンサとまとめてセンサ・アクチュエータと表現する。より厳密には LED などはインジケータと呼ばれる。

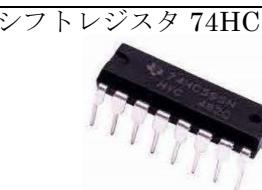
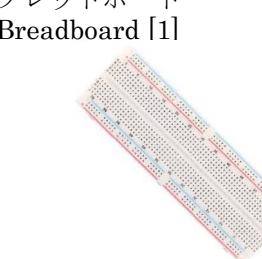
2.7.3. 抵抗・コンデンサ

回路を設計するため、次の抵抗・コンデンサを扱う。

 <p>抵抗セット Resistor [120] 10Ω、100Ω、220Ω、330Ω、1KΩ、2KΩ、5KΩ、10KΩ、100KΩ、1MΩの抵抗がある。抵抗値はカラーバーを使って確認すること。</p>	 <p>電解コンデンサ Electrolytic Capacitor (100 μ F 50V[2] · 10 μ F 50V[2]) 比較的大容量であるが、Tpd が長く極性があるので注意する。</p>	 <p>セラミックコンデンサ Ceramic Capacitor (22pF[5] · 104pF[5]) 極性がない。</p>
--	---	---

2.7.4. 電源・配線・その他デバイス

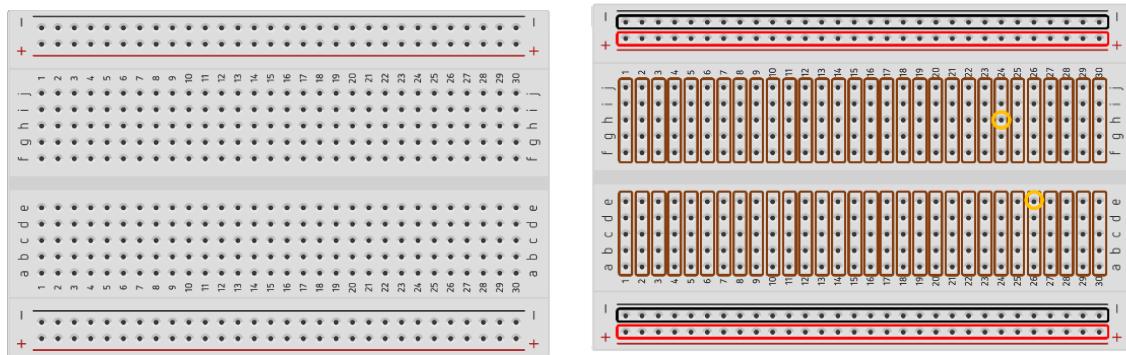
回路を設計するにあたり、次のようなデバイスや配線などを利用する。

 <p>整流ダイオード Diode Rectifier (1N4007) [5]</p>	 <p>トランジスタ NPN Transistor (S8050, PN2222) [5]</p> <p>複数の LED をデジタル出力の digitalWrite(20mA)で点灯すると電流が不足する。この際の電流増幅などに利用する。</p>	 <p>電源供給モジュール Power Supply Module [1]</p> <p>9V 以上の電源印加は厳禁である。</p>
 <p>シフトレジスタ 74HC595 [1]</p> <p>8 ビットシフトレジスタで、シリアル入力とクロック入力により 8 ビットの出力が順番に変化する。</p>	 <p>プッシュ・プル 4 チャネルドライバ L2930 [1]</p> <p>高い電力を制御する際に用いる。リレーよりも高速に動作し寿命が長いが、リレー程大電力を制御できない。</p>	 <p>9V バッテリ 9V Battery With DC [1]</p> <p>Arduino を電池駆動化する。</p>
 <p>ブレッドボード Breadboard [1]</p>	 <p>ジャンパ線 · DuPont Wire [各 10]</p>	 <p>USB ケーブル [1]</p> <p>Arduino と PC を接続する。</p>

特に、ジャンパ線とブレッドボードについては簡単に説明を加える。

- ジャンパ線はコネクタ線・デュポンワイヤとも呼ばれ、オス・メスの形状を持つピンとコネクタが両端に取り付けられたケーブルである。配線の便宜から様々なカラーリングがされている。一般には、黒は 0V・グラウンド・GND 配線に利用し、赤は 5V(3.3V)・電源・Vcc 配線に利用し、これらは固定電圧かつ電源供給用である。その他の色は各種信号線に利用する。当実験では、回路の設計ミスを避け、回路の確認の便宜を図るために必ずこの配線色のルールに従うこと。

- ブレッドボードは半田付けを行わずに電子回路を作成するための道具として広く復旧している。ブレッドボードに電子部品を差し、ジャンパ線を用いて電子部品や Arduino を接続できる。典型的なブレッドボードは次のような形を有する。ブレッドボードには通常、赤と黒もしくは青で示された+とーの列があり、これらは同一列であればすべて内部で接続されている。異なる+やーの列は相互に絶縁されている。また、a から e の列(この場合行とも言えるが縦横の区別がないため列とする)は、同一番号の列について内部でコネクタが接続されており、f から j の列も接続されている。すべての接続の形を示した図も示しておく。コネクタの場所は行と列の名称で指定され、例えば、図の黄色のコネクタ 2 つを結ぶ場合は、「24h と 26e を結線する」といった表現となる。配線は 5 組まで接続でき、それ以上は別の列に配線して拡張する。



2.7.5. Arduino と ZigBee シールド

Arduino の機能拡張を容易にするために特別に設計された拡張ボードをシールドと呼ぶ。Arduino のピン配置や基板のサイズに合うように設計されており、Arduino の上に重ねて利用できる。一般に比較的複雑な機能やピン数の多いデバイスの接続に利用されることが多く、信号線の接続を簡単にすることができる。さらに、電源も Arduino から供給するように設計されている場合が多い。シールドの例として、無線デバイスや、多チャネル電源リレー、ディスプレイなど多様なシールドが存在する。本実験では、ZigBee シールドを利用して Arduino を無線化する。Arduino 本体と ZigBee シールド間の情報交換にはシリアル通信を用い、専用のコマンドパケットを用いて制御する。

3. 言語仕様

Arduino は C/C++を基本としつつ独自に拡張された言語、Arduino 言語を利用して設計する。C 言語のすべての文法項目を利用でき、さらに C++の一部機能をサポートしている。初心者でも設計しやすいように、Arduino の基本的な利用形態を想定した様々なライブラリや関数が揃っているため、通常の処理は準備された関数を呼び出すだけで利用できる。一方で、機能それぞれに関数が備わっているため、関数の数は必然的に多い。覚えるよりも、調べて利用できること。

3.1. Python との違い

Python を習得済みであるため、その差異から学ぶのが近道であろう。まず、関数は類似している部分もあるが基本的に習得しなおす必要がある。web において様々な情報が提供されており、マニュアルも丁寧に記述されているため、調べながら学ぶ上で障害はないであろう。

まず、次の点に注意する。

- import で必要なライブラリを読み込むのではなく、#include <>で読み込む。
- フリーフォーマットのため、各文の最後にセミコロン（;）が必ず必要である。

- Python のようにタブでブロックを構成せず、{}でブロックを構成する。タブには意味がない。
- 変数はすべて型宣言が必要である。

なお、C 言語同様メモリ管理を自身で行う必要があるが、malloc などを利用することは稀である。これは、ヒープメモリを利用するとメモリ管理のために余計なメモリを追加で利用するため、RAM のサイズが小さいことからメモリの使用効率の悪さが問題となるためである。とはいっても、必要に応じて使わなければならぬケースや、C 言語取得において最初の関門であるポインタについて理解しなければならないケースも存在する。

3.2. 基本構造

`setup()` と `loop()` という 2 つの関数で構成され、`setup()` は Arduino ボードの電源を入れたときやリセットしたときに、一度だけ実行される。変数やピンモードの初期化、ライブラリの準備などに利用する。`loop()` は実際に実行するプログラムを記述し、電源が投入されている間無限ループとして実行され続ける。`setup()` や `loop()` は省略できない。

以下にプログラミングに用いることができる構造制御文および演算子を示す。既習の Python と記述スタイルは異なるが、基本的な動作は同一であるため、詳細は省略する。

3.3. 制御文

if, if else	条件制御文である。 一般的な動作を行うため説明は省略する。	<pre>if(条件文){ // 条件に該当するとき実行 } if (a < 500) { // 動作A } else if (a >= 1000) { // 動作B } else { // 動作C }</pre>
switch case	条件制御文である。 一般的な動作を行うため説明は省略する。	<pre>switch (var) { case 1: // varが1のとき実行 break; case 2: // varが2のとき実行 break; default:// (省略可能) // 不一致のとき実行 }</pre>
for	繰り返し制御文である。for 文は Python と仕様が異なる。	<pre>for(初期化; 条件式; 加算) { // 実行される文; }</pre>
while	繰り返し制御文である。while は Python と同様の仕様である。	<pre>while(条件式){ // 実行される文 }</pre>
do while	繰り返し制御文である。do while は後判定ループ構造記述ステートメントであり Python には存在しない。Python はなるべく記述多様性を無くすように言語仕様を限定しているが、Arduino 言語は C 言語と同じく、多様性を与え記述性を高めるように設計されている。	<pre>do{ // 実行される文 } while(条件式);</pre>
break continue	実行順序制御文である。Break はブロック離脱、continue は再実行を行う。	
return goto	return は関数から戻り、戻り値の追加もできる。goto は対応するラベル(文字列)へ実行を移す	

3.4. コメント

C++と同様、//および/* */でコメントを記述できる。

3.5. 演算子

以下の演算子があるが、すべて標準プログラム言語に準拠するため説明を省略する。

算術演算子	加算+ 減算- 乗算* 除算/ 剰余%が利用できる。
代入演算子	=を利用して変数に値を代入する。
比較演算子	==, !=, <, >, <=, >=が利用できる。真の時 1、偽のとき 0 となる。
ブール演算子	論理積 && 論理和 否定 ! が利用できる。
ビット演算子	ビット演算子は変数をビットのレベルで計算する。AND &, OR XOR ^ NOT ~ 左シフト << 右シフト >>が利用できる。
sizeof 演算子	変数や配列のバイト数を返す。

3.6. データ型

変数を宣言する時、データ型つまり型を必ず指定しなければならない。次のデータ型が利用できる。

なお、組み込み系マイコンのため、同じサイズでも異なる表現があり、型のバリエーションは少ない。

boolean	ブール型は true か false どちらかの値を持つ。true は 1、false は 0 として扱われる。
char	固定長 1 バイト、つまり 1 文字を記憶する型である。文字は'A'のように、シングルクオーテーションで囲って表記する。文字であるが、実際にはアスキーコードによる数値として記憶される。なお、char は正負が表現できる点に注意する。値の範囲は -128 から 127 までである。
byte unsigned char	符号無しのデータ型で 1 バイトの型。正の数のみ表現できる。char 同様文字も表現できる。値の範囲は 0 から 255 までである。
int	整数型と呼ばれ、頻繁に利用される数値の型である。実験で用いる Arduino は 2 バイト利用するため、値の範囲は -32768 から 32767 までである。
unsigned int word	2 バイトの値を格納する点で int 型と同じであるが、正の数のみ表現できる。値の範囲は 0 から 65535 まで。
word	符号無し 16 ビットのデータで、値の範囲は 0 から 65535 までである。
long	(long 整数型)4 バイト表現で、値の範囲は -2,147,483,648 から 2,147,483,647 まで。
unsigned long	(符号なし long 整数型)4 バイト表現で正数のみ表現できる。値の範囲は 0 から 4,294,967,295 まで。
float double	浮動小数点型であり、小数を扱う場合に利用する。4 バイトで値の範囲は 3.4028235E+38 から -3.4028235E+38 まで。実行速度が整数の変数よりも劣るため、必要なところでのみ利用する。表現誤差や計算誤差が発生し、思い通りの動作とならない場合があるので注意すること。なお、Arduino は float も double も精度は同じである点に注意する。
void	型がないことを意味する型である。Arduino では関数定義にのみ利用され、関数に戻り値がないことを示す。なお、setup や loop 関数は、void setup(){}、void loop(){} である。

- 文字列(配列) : C 言語では文字列は文字の配列として扱うが、これにはメモリ管理を含む少々複雑な表現と知識が求められる。特に C 言語ではポインタの利用が必須となるため、初心者にとって扱いやすいとはいえない。Arduino 言語も文字の配列として文字列を表現しており、C 言語と変わらない。一方で C++では同様に文字の配列として文字列を表現するが、String クラスが定義されており、比較的平易に文字列を扱うことができる(後述)。Arduino 言語においても標準で文字列型である String が利用できる。
- 配列 : 変数の集まりで、インデックス番号(添え字)を使ってアクセスすることができる。

3.7. String

`String str = "Experiment";` といった具合に宣言することで利用できる。その他、様々な初期化方法が定義されているため、各自で調査すること。`String` には文字列操作関数が豊富に用意されており、記述しやすく、またプログラムの見通しがよくなる。次のようなメンバ関数が用意されている。

<code>str.charAt(n)</code>	文字列の先頭から <code>n+1</code> 番目の文字を返す。
<code>str.compareTo(str2)</code>	<code>str</code> と <code>str2</code> の 2 つの文字列を比較する。辞書順で <code>str2</code> が後ろならば負、前ならば正の値を返す。一致するときは 0 を返す。
<code>str.concat(str2)</code>	<code>str</code> の末尾に <code>str2</code> が連結される。
<code>str.endsWith(str2)</code>	<code>str</code> の末尾が <code>str2</code> のとき <code>true</code> を、そうでなければ <code>false</code> を返す。
<code>str.equals(str2)</code>	2 つの文字列を比較し、一致する場合は <code>true</code> 、そうでなければ <code>false</code> を返す。
<code>str.equalsIgnoreCase(str2)</code>	<code>String.equals</code> と同じ動作であるが、比較において大文字小文字を区別しない。
<code>str.getBytes(buf, len)</code>	文字列を <code>byte</code> 型の配列(<code>buf</code>)にコピーする。 <code>len</code> は <code>buf</code> のサイズを表す(int)。
<code>str.indexOf(val, from)</code>	文字列の中を先頭から検索し、見つかった場合はその位置を返す(1 文字目を 0 と数える)。存在しないときは-1 を返す。 <code>val</code> は探したい文字または文字列で、 <code>from</code> は検索を始める位置を表す。 <code>from</code> は省略可能である。
<code>str.lastIndexOf(val, from)</code>	<code>indexOf</code> と同様であるが、文字列の中を末尾から検索する。
<code>str.length()</code>	文字列の長さ(文字数)を返す。
<code>str.replace(substr1, substr2)</code>	<code>substr1</code> を <code>substr2</code> に置換した文字列を返す。
<code>str.setCharAt(index, c)</code>	<code>index</code> で指定した位置の文字を <code>c</code> に置き換える。 <code>str</code> の長さより大きい <code>index</code> を指定した場合は何も実行されない。
<code>str.startsWith(str2)</code>	<code>str</code> の先頭が <code>str2</code> のとき <code>true</code> を返し、そうでなければ <code>false</code> を返す。
<code>str.substr(from, to)</code>	文字列の一部分を返す。 <code>to</code> は省略可能で、 <code>from</code> だけが指定されているときは、 <code>from+1</code> 文字目から末尾までの文字列を返す。 <code>to</code> が指定されている場合は、 <code>to</code> までを返す。
<code>str.toCharArray(buf, len)</code>	文字列を <code>byte</code> 型の配列(<code>buf</code>)にコピーする。 <code>len</code> は <code>buf</code> のサイズを指定する。
<code>str.toLowerCase()</code>	大文字を小文字に変換する。
<code>str.toUpperCase()</code>	小文字を大文字に変換する。
<code>str.trim()</code>	先頭と末尾のスペースを取り除く。

3.8. 定数

以下の定数がある。組み込み設計に特徴的な定数が備わっている。

<code>true/false</code>	論理レベルを定義する。標準プログラム言語に準拠するため説明を省略する。
<code>HIGH/LOW</code>	ピンの電圧レベルを定義する。なお、1 と <code>true</code> と <code>HIGH</code> は等しく、0 と <code>false</code> と <code>LOW</code> は等しい。電圧レベルでは、 <code>HIGH</code> は 5V、 <code>LOW</code> は 0V を意味する。
<code>INPUT/OUTPUT</code>	デジタルピンを定義する。その名前の通り、入力は <code>INPUT</code> 、出力は <code>OUTPUT</code> で指定する。
整数の定数	<ul style="list-style-type: none"> 十進数は特に指定なく 123 などと表記する。 二進数は B1111011 のように大文字の B に続く 0 と 1 の文字や文字列で表記する。 八進数は 0173 のように先頭の 0 に続いて 0-7 の文字や文字列で表記する。 十六進数は 0x7B のように先頭の 0x に続いて、0-9 および A-F や a-f で表記する。 その他、U フォーマッタと L フォーマッタがあり、u および U は 33u などのように用いて符号無しの数を表す。l および L は、100000L のように用いて倍精度の数を表す。Ul および UL は倍精度・符号無しを意味する。
浮動小数点数の定数	<code>E</code> や <code>e</code> が指数記号として利用できる。標準プログラム言語に準拠するため説明を省略する。

3.9. 変数

変数のスコープは標準プログラム言語に準拠するため説明を省略する。`cast` による型変換や暗黙の型変換が行なわれる。変数は宣言時に `static`、`volatile`、`const` の修飾が可能である。

<code>static</code>	関数内で <code>static</code> 宣言された変数はスタティック変数と呼ばれ、 <code>static</code> を付けない変数はローカル変数と呼ばれる。ローカル変数は関数が呼ばれるたびに生成と破棄が行われるのにに対し、スタティック変数は持続的で、値が保存される。スタティック変数は、関数が初めて呼ばれたときに 1 度だけ生成もしくは初期化される。
<code>volatile</code>	型宣言の前に付けて、コンパイラが変数をレジスタではなく RAM からロードするように指示する。 <code>volatile</code> は変数が割り込みや並列動作において外部から変更される可能性があるときに用いられ、Arduino では割り込みサービスルーチンで用いられる。
<code>const</code>	変数ではなく定数を表し、変数を変更不可にする。

3.10. 関数

様々な関数が準備されており、すべてを説明することは困難かつ意味がない。適宜調べて利用することが望ましい。以下、代表的な関数について簡単な動作説明とともに列挙する。

入出力定義関数	
pinMode(pin, mode)	ピンの動作を入力か出力に設定する。INPUT_PULLUP を指定することで、内部プルアップ抵抗を有効にできる。
デジタル入出力関数	
digitalWrite(pin, val)	HIGH または LOW の値(val)を、指定したピンに出力する。指定したピンが pinMode()関数で OUTPUT に設定されているべきである。
digitalRead(pin)	指定したピンの値を HIGH か LOW で戻り値として読み取る。
アナログ入出力関数	
analogRead(pin)	Arduino は 6 チャネルの 10 ビット AD コンバータを搭載しており、指定したアナログピンから 0 から 5 ボルトの入力電圧を 0 から 1023 の数値でアナログ値を戻り値として読み取る。読み取りに約 100μ秒(0.0001 秒)かかる。
analogWrite(pin, val)	指定したピンから PWM を出力する。正確にはアナログではない。
analogReference(type)	アナログ入力で使われる基準電圧を設定する。analogRead 関数は入力が基準電圧と同じとき 1023 を返す。Type は DEFAULT: 電源電圧(5V)を基準電圧とする(デフォルト)、INTERNAL: 内蔵基準電圧を利用する、EXTERNAL: AREF ピンに供給される電圧(0V~5V)を基準電圧とする、から選択できる。
その他の入出力関数	
shiftOut(dataPin, clockPin, bitOrder, val)	マルチプレクス動作を行う。dataPin と clockPin が出力。スタートビットを最上位ビット(MSB)、最下位ビット(LSB)どちらでも指定できる。各ビットは dataPin から、clockPin の反転と同期して出力される。bitOrder は MSBFIRST または LSBFIRST を指定する。
shiftIn(dataPin, clockPin, bitOrder)	デマルチプレクス動作を行う。clockPin を HIGH、dataPin から 1 ビット読み込み、clockPin を LOW の状態を繰り返す。戻り値は読み取った値(byte)となる。クロックの立ち上がりエッジで読み取る場合は、事前に digitalWrite(clockPin, LOW)としてピンを LOW にしておく必要がある。
pulseIn(pin, val, timeout)	ピンに入力されるパルスを検出する。入力が HIGH から LOW となる間の時間(パルス長)をマイクロ秒単位で戻り値(unsigned long)として返す。タイムアウトを指定しその時間を超えた場合は 0 を返す。計測可能な時間は 10 マイクロ秒から 3 分である。
tone(pin, f) tone(pin, f, duration)	指定した周波数 f の矩形波(50% duty)を生成する。duration で指定した時間矩形波を発する。duration を指定しなかった場合は noTone()を実行するまで動作を続ける。同時に生成できる矩形波は 1 音のみである。tone()実行後他のピンで tone()を実行する際には、事前に noTone()を実行しておかなければならない。また、31Hz 以下の矩形波は生成できない。tone を利用するとピン 3 および 11 から PWM 出力ができなくなる。
noTone(pin)	tone()で開始された矩形波の生成を停止する。
時間に関する関数	
millis()	Arduino の実行開始時からの時間をミリ秒で戻り値(unsigned long)として返す。約 50 日間でラップラウンドする。
micros()	同様にマイクロ秒単位で戻り値として返す。約 70 分間でラップラウンドする。
delay(ms)	プログラムを指定した時間(ミリ秒指定)だけ止める。
delayMicroseconds(us)	プログラムを指定した時間(マイクロ秒指定)一時停止する。
数学的な関数	
min(x, y), max(x, y), abs(x)	最小値、最大値、絶対値を与える
constrain(x, a, b)	x: 計算対象の値、a: 範囲の下限、b: 範囲の上限として数値を指定した範囲に換算する。 $a \leq x \leq b$ では換算値が、 $x < a$ では a、 $x > b$ では b が戻り値となる。
map(val, fromLow, fromHigh, toLow, toHigh)	数値のある範囲から別の範囲に変換する。fromLow が toLow に、fromHigh が toHigh になる。その間は範囲の比に基づく値になる。範囲外であっても切り捨ては行われない。整数のみ扱い、小数部分は切り捨てられる。戻り値は変換後の数値(long)である。
pow(base, exponent)	base: 底となる数値(float)、exponent: 指数となる数値(float)としたべき乗を戻り値(double)とする。
sqrt(x)	平方根を戻り値(double)とする。
sin(rad), cos(rad), tan(rad)	正弦、余弦、正接を戻り値(double)とする。ラジアン(float)で指定する。
randomSeed(seed) random(min, max)	randomSeed で long の値による乱数の種を与え、random で生成する乱数の下限(省略可)と上限を指定して min から max-1 の間の疑似乱数整数 (long)を戻り値とする。
lowByte(x), highByte(x) bitRead(x, n), bitWrite()	x の下位/上位 1 バイトを取得して戻り値(byte)とする。

bitSet(x, n), bitClear() bit(n)	x の LSB から n ビット目を取得/設定する。bitRead の戻り値は 0 または 1 となる。 x の n ビット目をセット/クリアする。 n ビット目のみ 1 にした値を戻り値とする。
割り込み	
attachInterrupt(interrupt, function, mode)	pin2(int.0) pin3(int.1)の外部割込み発生時に実行する、引数戻り値とともに void の関数 function を指定し、interrupt で割込み番号を指定する。mode でトリガ条件を LOW(ピンが LOW)、CHANGE(ピンが変化)、RISING (posedge 変化)、FALLING (negedge 変化) から選択する。注意点として、delay 関数や millis 関数は機能せず、シリアル通信の受信が不安定になる可能性がある。割込み関数内で変数は volatile 型とすること。
detachInterrupt(interrupt)	interrupt で指定した割り込みを停止する。
interrupts(), noInterrupts()	割込みを有効/無効にする。
シリアル通信	
Serial.begin(speed)	シリアル通信のデータ転送レート(bps/baud)を指定する。ここでは 9600 を指定する。
Serial.end()	シリアル通信を終了する。再開は Serial.begin()を呼び出す。
Serial.available()	シリアルバッファにあるデータのバイト数を返す。バッファは 64 バイトまで。
Serial.read()	受信データをバイト毎読み込む。戻り値が -1 の場合データは存在しない。
Serial.peek()	受信データをバイトで読み込むが、バッファ中の読み取り位置は維持される。
Serial.flush()	データの送信がすべて完了するまで待つ(強制排出)。
Serial.print(data, format) Serial.println(data, format)	ASCII テキストかつ指定された形式でデータをシリアルポートへ出力する。浮動小数点数の場合は、小数点以下第 2 位までデフォルトで出力される。バイト型のデータは 1 文字として送信されます。文字列はそのまま送信されます。println は改行コードが最後に送信される。詳細は各自で調べること。
Serial.write(val) Serial.write(str) Serial.write(buf, len)	シリアルポートにバイナリデータを出力する。val: 送信する値(1 バイト)、str: 文字列(複数バイト)、buf: 配列として定義された複数のバイト、len: 配列の長さが指定できる。戻り値は送信したバイト数 (byte)である。

3.11. シリアル通信

他のコンピュータやデバイスとの通信に利用する。ピン 0 と 1 がシリアルポートのピンで、この 2 ピンを通信に使用する場合、デジタル入出力として使うことはできない。また、Arduino IDE はシリアルモニタを備えており、Arduino と直接通信できる。なお、shiftIn()はソフトウェア実装されたシリアル入力関数であり、どのピンでも利用できる。これらはシリアル信号の入力開始の仕方が異なっており、Serial クラスのシリアル通信は RS-232C などの規格で定められた一般的なシリアル通信を扱うことができる。

shiftIn()はソフトウェア実装のため他の処理を行っている間は利用できず、実際には割り込みなどを併用する必要がある。また 74HC165 などのデバイスから入力されるシリアル入力を読み込むと仕様上最初のビットの読み込みができないためにビットがずれるという現象が発生する。シリアル入出力は、digitalRead などを用いることで独自仕様でも設計可能であり、対象となるプロトコルに合わせて個別実装が求められる場合もある。

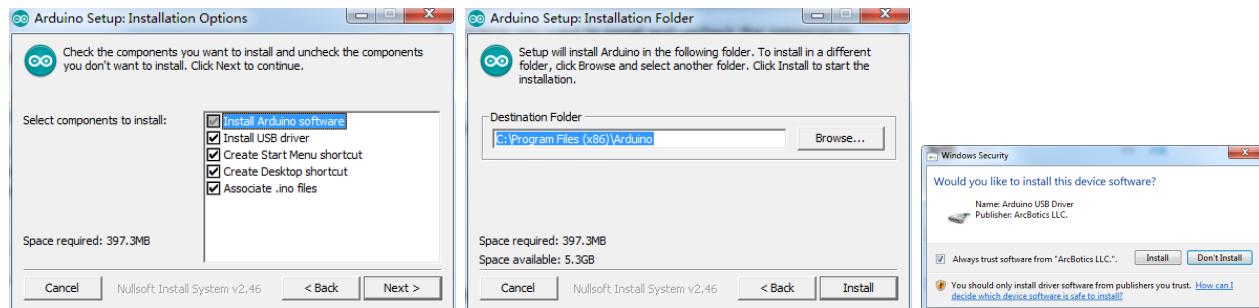
3.12. 標準ライブラリと使い方

Arduino のライブラリは、あらかじめ IDE に付属する標準のライブラリと、コミュニティーから寄稿されたライブラリの 2 種類がある。寄稿されたものはユーザが個々にダウンロードしてインストールする必要がある。標準ライブラリとして、例えば EEPROM(外部メモリ)、SoftwareSerial(シリアル通信)、Stepper(ステップモータ制御ライブラリ)、Wire(2 線インタフェースである TWI や I2C を用いた通信)、SPI(Serial Peripheral Interface バスを用いた通信)、Servo(サーボモータ制御)、LiquidCrystal(LCD パネル表示)、Ethernet、SD メモリ、タイマなど、様々提供されている。このライブラリの豊富さも Arduino の魅力であり、簡単に制御プログラムを構築できる。

4. 実験準備

4.1. Arduino IDE のインストール

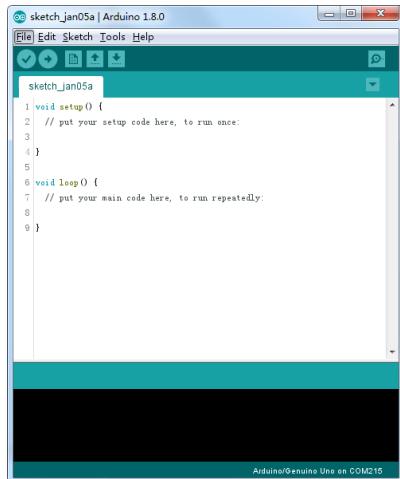
Arduino IDE のダウンロードサイト(<https://www.arduino.cc/en/software>)からダウンロードしてインストールする。Windows、Mac、Linuxなどのプラットフォームで利用できるが、インストール手順はプラットフォームにより若干異なるため、各自で確認してインストールする。Windowsでは最新バージョンのexeファイルをダウンロードし、インストーラを用いてインストールすると便利である。Arduino USB ドライバもインストールすること。

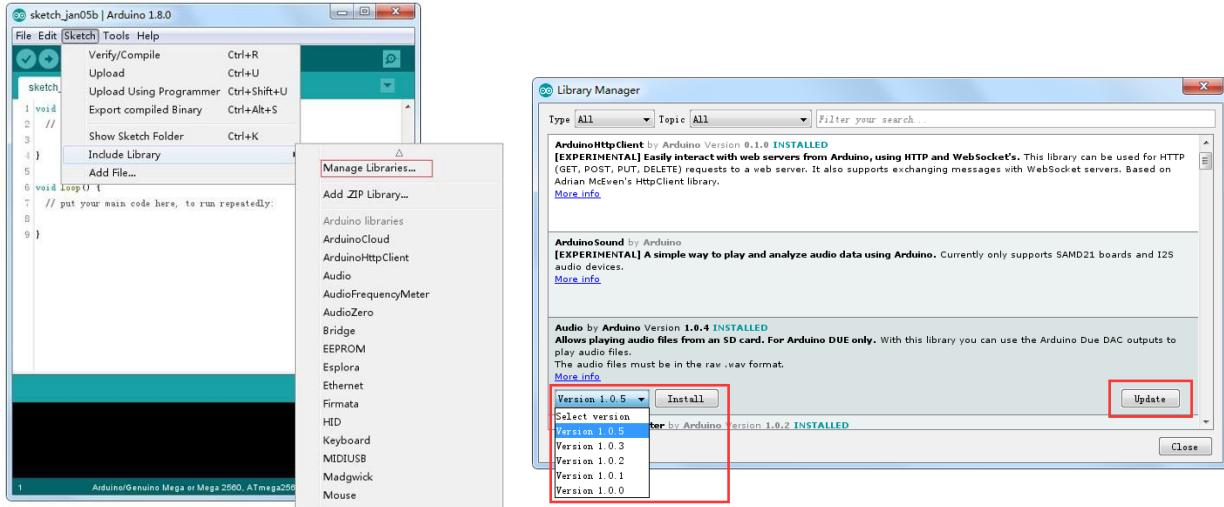


インストールすると、Arduino のアイコン  がデスクトップに配置される。これをクリックして Arduino IDE を起動することができる。起動を確認したら、Arduino を PC に USB ケーブルを用いて接続する。Arduinoへの電源供給も同時に行われるため、接続するだけで認識また、プログラムの書き込みが可能となる。ただし、初回利用時はドライバのインストールが必要となる場合があるため、必要に応じてドライバをインストールする。

4.2. ライブラリマネージャ

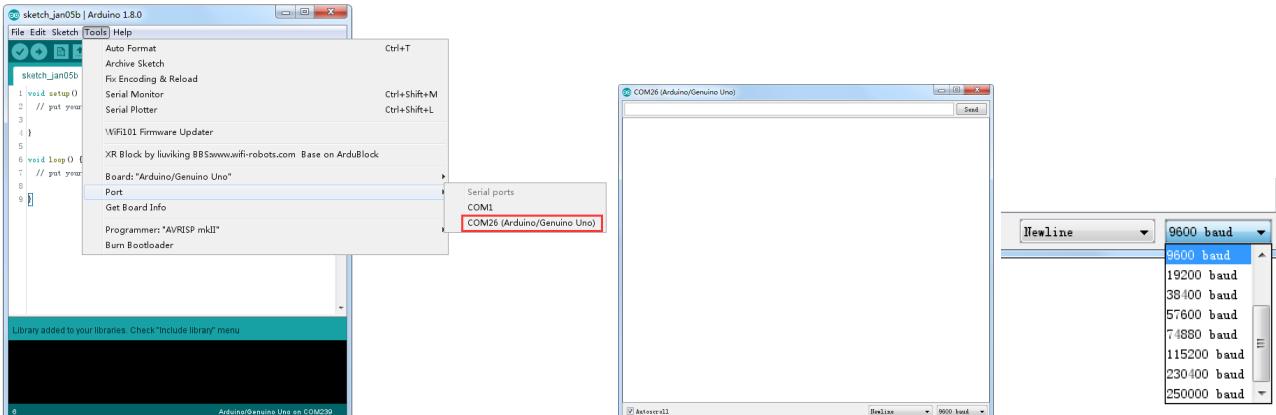
Arduino にセンサやアクチュエータを接続して利用する際、複雑なプログラム設計が必要となるが、専用のライブラリを Arduino IDE に導入することでこれを回避できる場合がある。利用するには、「スケッチ」メニューから「ライブラリ」の「ライブラリを管理」で開く「ライブラリマネージャ」を利用する。ライブラリのリストから導入したいライブラリを選び、必要に応じてバージョンを選択してインストールする。また、zip ファイルでライブラリが提供される場合は、「ライブラリ」から「ZIP ライブラリを追加」を選択してインストールする。この zip ファイルは Arduino スケッチディレクトリの libraries フォルダに展開される。また、ライブラリを手動インストールすることも可能である。各マニュアルを見て対応すること。





4.3. シリアルモニタ

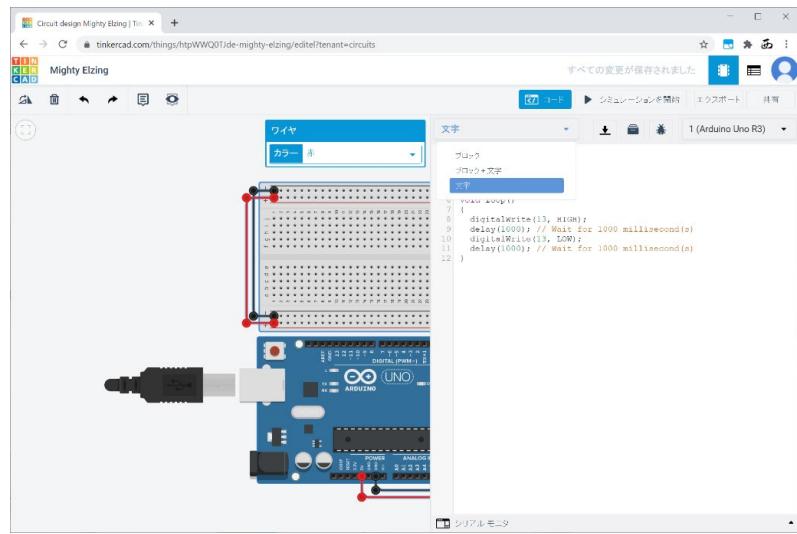
Arduino IDE のシリアルモニタを利用して、動作中の Arduino から USB シリアルを経由してメッセージを表示させたり、デバッグコード等を用いて動作状況を確認したりすることができる。シリアルモニタアイコンを選択して利用する。なお、シリアルを利用するため、シリアルポートとボーレートが一致しなければならない。「ツール」の「シリアルポート」よりポートを指定し、シリアルモニタ右下のプルダウンメニューからコードで指定したのと同じボーレートを選択する。左下隅の自動スクロールチェックボックスにより画面から文字があふれた場合に自動スクロールするかどうかを制御できる。



4.4. Arduino シミュレータ

予習や復習において、Arduino や周辺キットを利用した学習を行うことは困難であろう。実物はなくとも、シミュレータにより仮想実験を行うことができ、さらに Web を用いることで、いつでも、どこでも仮想実験を行うことができる。そのような環境の一つに Tinkercad.com Arduino Simulator (TAS) がある。TAS は Autocad により提供されており、公式サイト(<https://www.tinkercad.com/>)にアクセスしアカウントを作成することで利用可能である。TAS は 3D モデル設計環境も提供しており、最初に利用する場合は、「新しい回路」を選択して回路設計を開始する。既に設計した回路を利用する場合はその回路を選択することで、再度設計可能となる。

新しい回路を設計する場合、スターターから Arduino を選択し、本実験の環境に合わせるのであれば一番上のブレッドボードを選択して配置してから回路を設計するとよい。また、プログラム設計はブルック記述とテキスト記述が選択できるが、テキスト記述で行うこと。



5. 設計と実装

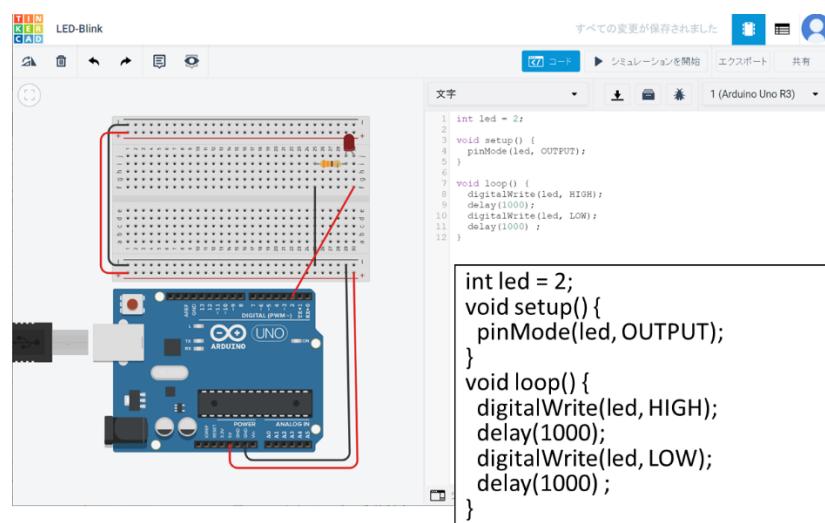
5.1. 実験を始めるにあたって

細かいパーツが数多くあり、また正しくケースに収めるには工夫がいるため、パーツを取り出す前に、何があるか、どのように入っているかをきちんと確認すること。スマートフォンで写真を取得しておくことをお勧めする。本実験では片づけの正確さ、清潔さを点数として評価し、加算する。紛失などしないよう、次に利用する側に立って丁寧に扱うこと。

5.2. 各種制御例

5.2.1. LED の自動明滅

まず基本的な回路およびプログラム設計として LED の明滅制御を行う。印刷の便宜から Tinkercad の画面を表示させるが、実際に回路を組み立てても当然ながら動作する。LED を直接接続すると過電流により LED に負担がかかり熱破壊する可能性があるため、必ず抵抗を直列に挿入する。抵抗の大きさで明るさが変化するが、キットに含まれている抵抗を用いる場合、実用的な抵抗値は 100Ω 、 220Ω 、 330Ω である。抵抗値の小さい方が明るい。ここでは、 220Ω で設計する。カラーコードは「赤赤茶」か「赤赤黒黒」で最後の色は許容誤差を示す。変数 led は LED を接続するピン番号を格納している。Setup では利用するピンの入出力を指定している。led のピンに電力を与えて LED を制御するため、Arduino からみて出力つまり OUTPUT となる。ここでは明滅させるため digitalWrite で HIGH つまり 5V を与えて点灯させ、delay(1000)で 1000ms(1 秒)待って digitalWrite で LOW つまり 0V として消灯させ、再び 1 秒待つという動作を loop で電源が入っている間繰り返す。なお、実機の Arduino では、一度プログラムすると、電源を遮断してもプログラムは消えず、電源を入れれば動作する。したがって、プログラムの際には PC は必要となるが、一度プログラムすれば、電源さえ供給されれば PC は不要となる。

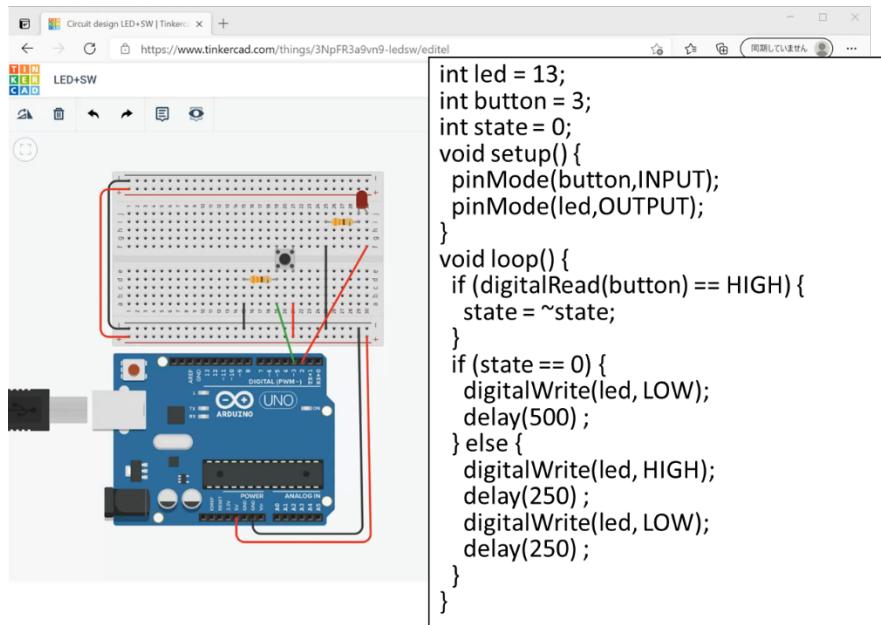


5.2.2. LED の外部信号制御

例えばある温度になったら LED が点滅するなどの応用を考える。この場合、外部からの信号入力にしたがって出力を制御することになるため、入出力が備わる。ここではシンプルな例としてボタンを入力として LED を制御する回路とプログラムを作成する。ボタンと LED を直列で結線した動作ではなく、ボタンを押すたびに明滅を繰り返す制御とする。

新たにボタン入力を行うピンを表す変数 button を定義し、3 番ピンを利用してボタン入力を行う。

loop 内では、まずボタンが押されているかどうかを判定、押されている場合は state を反転させる。state が 0 の時は消灯状態となり、state が 1 の時は点滅状態となる。この記述では digitalWrite が 500ms 程度で判定が繰り返されるため、ボタンを 0.5 秒程度押し続けなければ反応せず、長い間押し続けると消灯と点滅を繰り返す。この時間を短くすると敏感になりすぎ、チャタリングが発生する。長くするとボタンを押し続けなければ反応しなくなる。つまり、仕様は満たすが利用上問題のある記述である。解決には、より適切な実装である微分回路を模擬した差分判定とチャタリング防止遅延処理により posedge か negedge を検出すればよいが、ここでは省略している。



また、実際には INPUT ではなく、INPUT_PULLUP を利用することで、抵抗を用いずにスイッチの ON/OFF 検知が可能となる。入出力モードを INPUT_PULLUP にすることで Arduino 内部のプルアップ抵抗が利用されるため、外部のプルアップ抵抗を排除できる。例えば、

```
pinMode(button,INPUT_PULLUP);
```

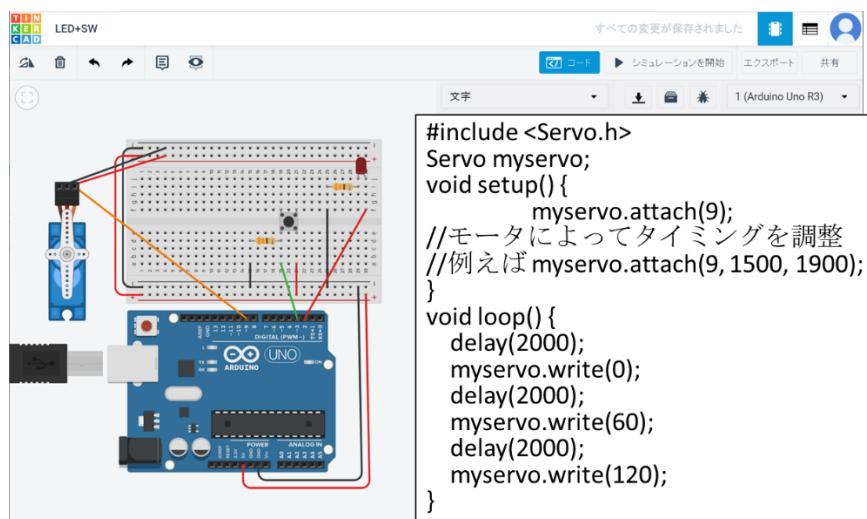
とすることで、マイコン内部の抵抗を介して電源レベルにつながるため、直接ボタンを繋ぎ、その先を GND につなぐだけでスイッチとして機能する。この場合は、Active Low の回路となるため、ボタンを押すと LOW、離すと HIGH になり、上記プログラムとは極性が逆になる。

5.2.3. サーボモータの制御

サーボモータとは、広義には制御できるモータの総称で、狭義では一般的な PWM 位置制御モータを指す。その動作形態から、指定した速度で回り続ける回転サーボモータと、指定した位置（角度）まで移動して停止する位置サーボモータがある。制御方法としては、位置サーボモータとしてラジコンなどで広く利用されている PWM 位置制御、回転サーボモータとして、モータードライバと DC モータを利用した制御、ステッピングモータを利用した制御がある。ここでは、PWM 位置制御サーボモータ(以降単にサーボモータ)を例に説明する。

サーボモータには 3 つの配線があり、赤色は電源、黒色や茶色は GND、それ以外の黄色やオレンジ色は制御信号の配線である。PWM 制御であり、0.5 から 2.4ms 程度を制御域として、可動範囲がおよそ 210 度、PWM 周波数は 20ms 程度である。なお、サーボモータのギア機構は壊れやすいため、手で

回してはいけない。また、電源と GND を間違える、無茶な角度指令値を入れるなどすると、モータに異常な負荷がかかり、壊れることがあるので細心の注意を払うこと。このような PWM 波形を直接コードを書いて制御してもよいが、PWM 生成に専念すると他の処理が滞る。そこで、専用のライブラリを利用する。専用ライブラリはタイマを利用して実装されており、他の処理も同時並行で行うことができる。実装例では、まず 2 秒待ってサーボモータの位置を 0 度にし、その 2 秒後に 60 度、さらに 2 秒後に 120 度にする動作を繰り返す。すなわち、2 秒毎に角度が変わる。Servo クラスを利用するため、`#include <Servo.h>`とする。Servo クラスのインスタンス myservo を定義し、myservo を用いてサーボモータの制御を行う。複数のサーボを利用する場合は、別のインスタンスを生成する。attach メンバ関数を用いて、サーボモータが接続されているピン番号を指定する。このピンから、PWM 信号が出力される。この際、PWM 波形のデューティーを指定できる。loop 内は、説明の通り、2 秒毎に 0 度、60 度、120 度と変化する。このように、比較的高度なデバイスを接続する際にも、ライブラリを用いることで簡単に利用することができる。



5.3. 実験環境のセットアップ

実験環境を構築する。既に必要なソフトウェアやドライバはインストール済みである。配布するスタートキットの内容物は下記写真を参照。最後に片スタートキットこの並びで片づけること。片づけが不十分な場合は、態度点を減点する。



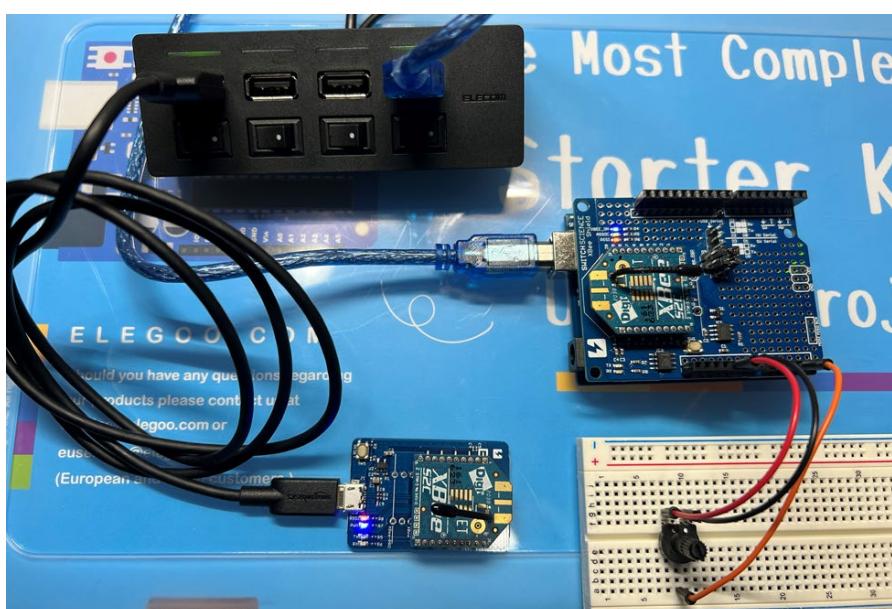
<ul style="list-style-type: none"> • Ir リモコン • Ir レシーバ • LCD 		<ul style="list-style-type: none"> • USB ハブ 			RFID キット 薄膜キーボード 抵抗キット パーツケース	
• ステッピングモータ • ステッピングモータドライバ	• 超音波モジュール • マイクモジュール • リレー	• 7Seg LED • 4×7Seg • ドットマトリックス LED	• ジョイスティック	• リアルタイムクロックモジュール	LED	電解コンデンサ RGB LED CdS
• サーボモータ	• XBee • XBee USB アダプタ	• Arduino 用バッテリ	• 温度センサ • 湿湿度センサ • ロータリーエンコーダ	• 人感センサ	白 LED	トランジスタ 整流ダイオード
• 水位センサ • DC モータ • プロペラ	• Arduino • USB ケーブル • オス・メスケーブル				ボリューム	スイッチ セラミックコンデンサ
					L293D 74HC595 サーミスタ	スピーカ×2 傾斜センサ

次に、PC の電源を入れて、指示通りログインする。その後、バッチファイル SDrefresh.bat を実行して実験フォルダを作成する。コマンドラインで Sdrefresh と入力するか、デスクトップ上の SDrefresh アイコンをクリックする。すべての作業は、ここで作成される D ドライブ batSD フォルダ下の実験日付のフォルダ内で行うこと。このフォルダの中で作成された内容のみ評価対象となる。

次に、USB ハブに電源を接続し、PC に接続する。電源の接続口がわかりにくいので注意すること。

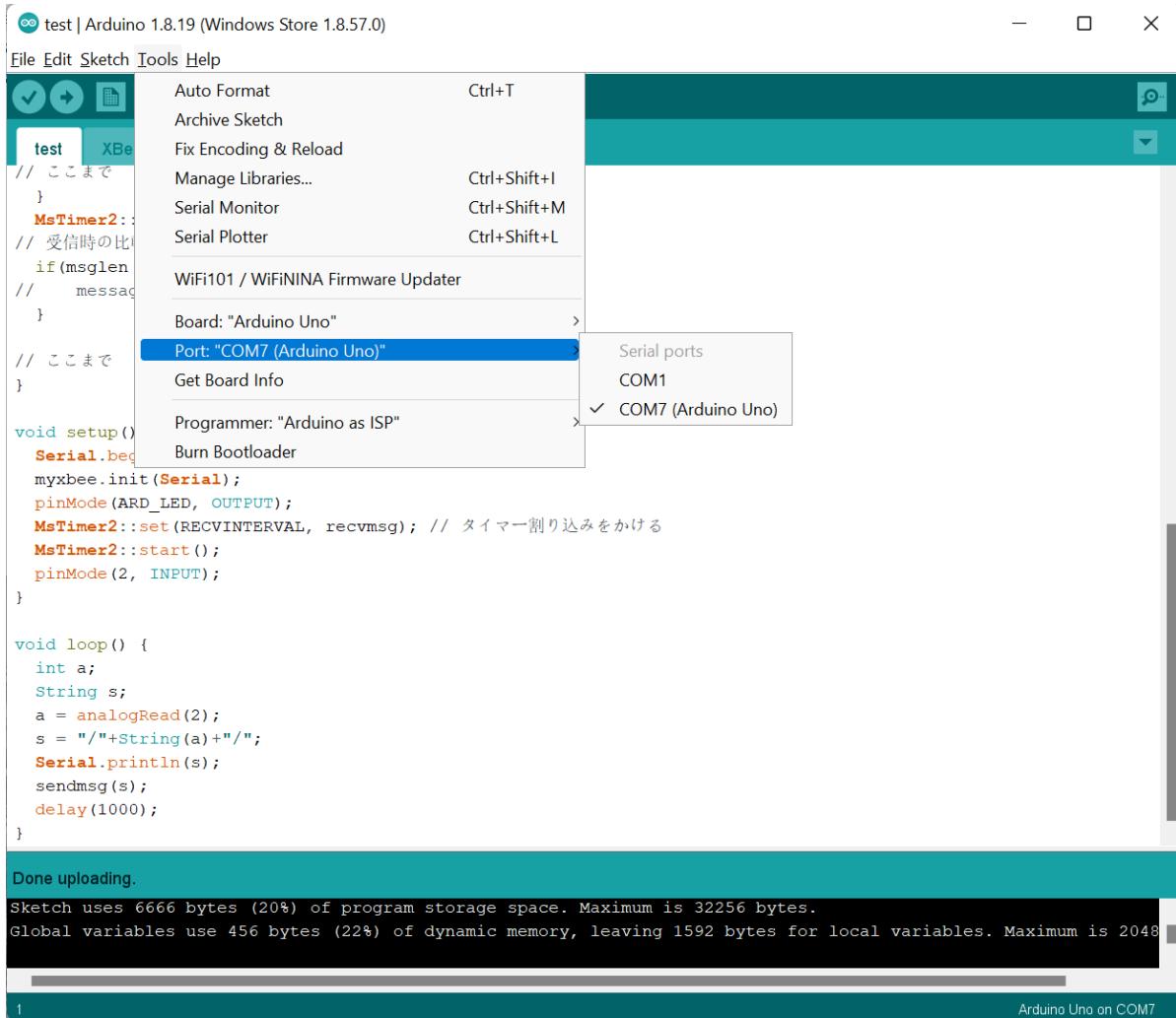
USB ハブに Arduino と XBee を搭載した XBee USB アダプタを接続する。USB ハブはスイッチ付きである。スイッチを ON にしてグリーンの LED が点灯するようにする。接続が正しければ、PC が認識して利用できるようになる。

次の図を参考に、可変抵抗(ボリューム抵抗)を接続する。両端を VCC と GND、真ん中を信号線で 2 番ピンに接続する。

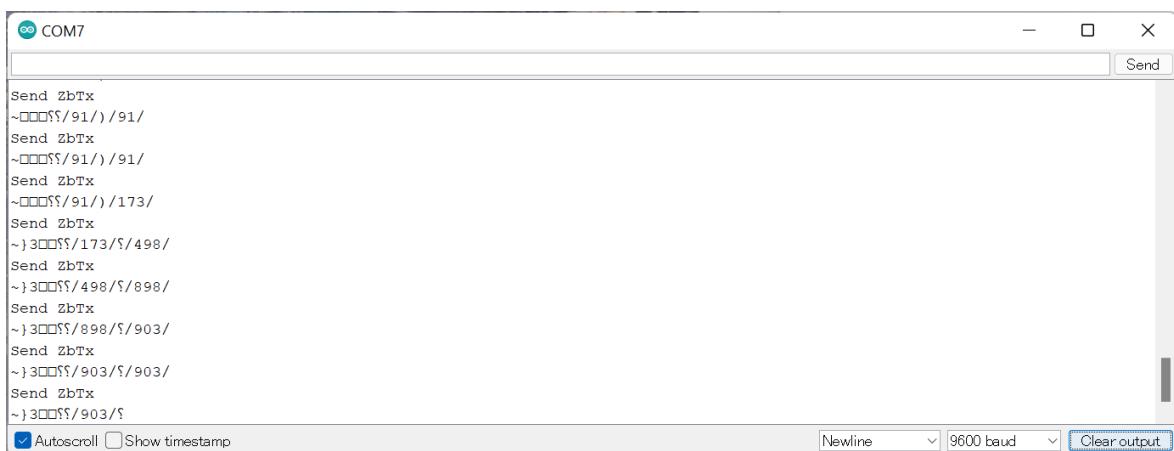


次に、配布する kit.zip を開く。kit は各自変更して壊してもよいようにコピーを保持することを推奨する。配布プログラムは、2 番ピンの電圧をアナログで読み取り、これを、スラッシュ"/"を前後につけて ZigBee で送信するように設計されている。

kit を改変なく Arduino に焼き込む。Tools の Board を Arduino Uno、Port も Arduino Uno にする。COM の番号は利用する PC の環境によって異なることに注意する。



→アイコンをクリックして、コンパイルと焼き込みを行う。次にシリアルコンソールを開いて、通信内焼き認する。



つまみを回すと、スラッシュで囲われた中の数字が、つまみの位置に併せて変化することから、つまみの位置がアナログで取得できていることがわかる。API エスケープモードを用いているため、前後に文字化けとも取れる XBee のコマンドが付与されている。

次に、XCTU のシリアルコンソールを開き、Open を押して Close にする。すると、ボーレートなどシリアルの設定が一致していれば、通信内容が表示される。この通信内容においても、同様にボリュームのつまみの値が表示されていることを確認する。スラッシュの間に数値が挟まれており、ボリュームを回すと、それに併せて変化することが見て取れるであろう。ここで表示されているボリューム抵抗の読みの数値は、XBee の無線を介して届いた値であり、無線通信が成功していることも同時に確認できる。この環境について、より詳しくみてみよう。

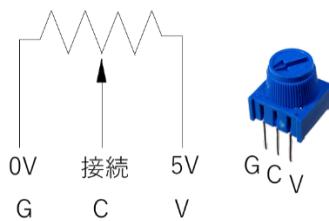
5.4. アナログデバイスとの接続

最も一般的なセンサの仕組みは、温度や光量、重さ、曲げ角度など、計測対象の変化により抵抗値も変化する素子を利用した対象変量の数値化である。ここで、その中でもシンプルかつ基本的なセンサとして、可変抵抗器の抵抗値つまり、つまみの位置を読み取る回路を設計する。可変抵抗を使って、ピンに与える「電圧」を変化させ、その電圧を Arduino 内蔵の A/D コンバータで読み取る。

Arduino には A/D コンバータが入っており、Analog と書かれた部分で読み取ることができる。Digital と記載されている 0 番から 5 番のピンは Analog と書かれた 0 番から 5 番と共に利用でき、どちらかしか利用できない。デジタルで利用するとアナログでは利用できず、また逆もしかりである。なお、実際には Arduino Uno の場合、アナログ入力ピンの 0~5 番をデジタル入出力ピンの 14~19 として利用できる。この時、マクロ定義の A0~A5 番を利用すると基板のシルク記載の通りに指定でき、次のような指定が可能である。

```
pinMode(A0,OUTPUT);
digitalWrite(A0,HIGH);
```

実際に可変抵抗値を読み取るには、まず次のように回路を構成する。C の電圧は、抵抗の内分比で決定され、つまみの位置によって定まる。そこで、C を Arduino のピンの接続し、AnalogRead で読み取ることで、可変抵抗はポテンショメータとして機能する。



```
int val; // 変数を定義
val = analogRead(ピン番号); // 抵抗の電圧を読み取り
```

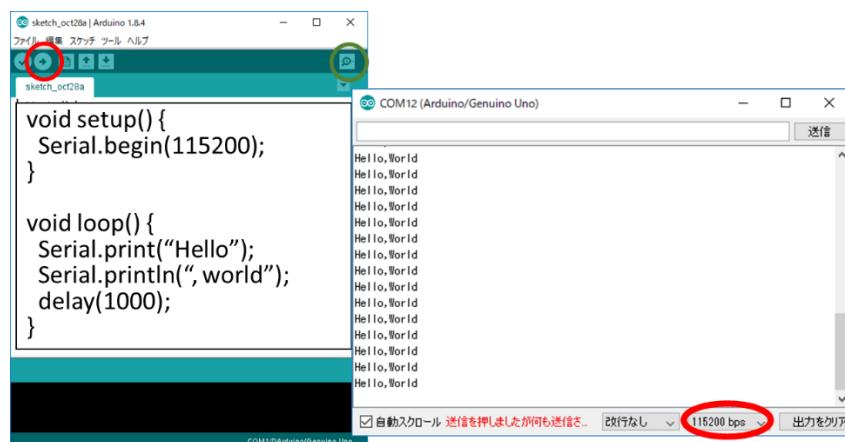
5.5. シリアル通信とデバッグ

配線を一本のみ用いて情報を送る通信方式であり、配線コストが低減でき広く普及している。また、その通信仕様も RS-232C に準拠した形で統一されている。さて、その仕様について簡単に見てみよう。例えば、1 と 0 をバタバタしただけでは正しく値が取得できない。お互いに、ボーレート（転送速度）を一致させて、一秒にいくつのデータを入れるのかを決めておく必要がある。例えば、1bps では 1 秒に 1 ビット送信するが、1 を送る場合、1 秒間 HIGH を維持する。これが、100bps で受け取ると、100 ビットの情報としてとらえて、1 が 100 個届いたと理解するであろう。もう一つ、ずっと 0 の場合、データが流れているのか、0 という有効なデータが流れているのかが判別できない。つまり、データの始まりがどこかが認識できなければならない。ノイズが載った場合誤った情報が伝わる可能性がある。このような問題について、スタートビットの利用や、パリティーによる誤り検出などの仕組みが提供されている。さらに、処理できないうちに次々にデータが送られてこないように、バックプレッシャー機能も提供されている。USB もユニバーサルシリアルバスという名前の通り、シリアル通信を基本として実装されている。

Arduino IDE とシリアル通信を行うことで、動作に必要な情報の通知や、デバッグメッセージの通知ができるようになり、利用の便宜が向上する。その例として Hello,World と表示する設計を行う。Arduino に直接液晶表示デバイスを繋いで画面にメッセージを出すことは可能であるが、デバイスを準備し接続するコストや設計するコストが伴う。デバッグなどは簡単に行なうことが求められるため、そのニーズを満たすことができない。そこで、Arduino と PC 間で直接通信することで実現する。ここまで Tinkercad を使って試すことができたが、シリアル通信は Arduino と PC をつなげる必要があるため、実機を用いる必要がある。

まず、手順として、次のプログラムを記述する。Setup では Serial.begin でボーレート(ここでは 115200)を指定する。loop では Serial.print()もしくは Serial.println()で PC に”Hello,World”を送って表示させる。1 秒の delay があるため、毎秒”Hello,World”と表示される。設計を終えたら、 を押してコンパイルを行う。エラーがなければそのまま書き込みが行われる。書き込みが行われたら を押してシリアルコンソールを開き、シリアルコンソールも同じボーレートに指定することで動作が確認できる。

シリアル通信は LED などとは比較にならない表現力があり、多くの情報を確認できる。コード中に Serial.print や println を配置して変数の値や実行場所、順序などを確認することで効率よくシンタックスではなくセマンティックなデバッグを進めることができるであろう。



5.6. ZigBee による無線通信

本実験で用いる ZigBee デバイスは、Full Function Device(FFD)である XBee S2C を利用する。ファームウェアの書換えにより、様々なプロトコル、機能に対応する。XBee は小型でありながら、直接センサを取り付けて計測値を ZigBee で通知することもできる多機能デバイスである。ここでは、シリアル通信を用いて Arduino と通信し、専用のコマンドを用いて Arduino から XBee を制御する。



XBee は Digi 社より提供されており、XBee の動作内容を変更するには、Digi のサイトよりダウンロードできる XCTU というツールを利用する。XBee には 2 つの通信モードがあり、Arduino で一般的なシリアル通信をそのまま無線化できるモードである AT モード (透過モード /Transparent mode) と、独自のデータ構造(API フレーム)を用いて通信を行う API モードがある。AT モードはお手軽であるが、機能が限定される。API モードは多少複雑な制御が必要であるが、高機能である。本実験では、API モードを利用する。

まず、XCTU をインストールし起動し、左上の+の「Add devices」もしくは虫眼鏡アイコンの「Discover devices」で XBee を認識させれば、XBee を検索できる。また、XCTU を用いてファームウェアの更新や機能変更も行うことができる。XBee の ZigBee には ID が振られており、配布している機器セット毎に異なる番号が振られており、他のセットの XBee デバイスが見えない、機器セット毎閉じたネットワークを構築するようにしている。XBee を扱うライブラリは、当研究室において独自に設計したライブラリであり、基本となる XBee.cpp と XBee.h で基本関数が定義されており、これらを組み合わせて、本実験で利用しやすく機能を集約した関数群が source.cpp と source.h で定義されている。XBee.cpp などはコードも長いため内容の詳細な説明は割愛するが、source.cpp についてはその内容の概要をぜひ押さえておいていただきたい。source.cpp は利用するアプリケーションにしたがって修正することで、よりニーズにあった通信ライブラリにすることができるであろう。その観点で、何かしら機能拡張する際のポイントについて説明する。

まず、XBee へあるデータを無線で通知するための関数は、参考プログラムにある sendmsg である。その中は、XBee クラスのメンバ関数である broadcastXBeeData を呼び出して、すべての XBee にパケットを投げることで、相手を指定せずに Arduino で計測した情報を配布している。全体ではなく特定の XBee モジュールにデータを送りたいときは、sendXBeeData メンバ関数を呼び出すとよい。このメンバ関数は、引数にアドレスとペイロードを指定できるため、アドレスの指定が可能である。

次に受信側について、XBee がメッセージを受信すると、このメッセージを Arduino に転送する必要がある。それには、シリアル通信を利用するが、パケットが来るたびに割り込みを掛けると、通信混雑時に、処理しなければならない情報が増える一方で、割り込みも増えて頻繁に通信データの受取を行うため、受け取った情報の処理が滞ることになる。つまり、通信量が増えると突然システムがダウンするといった状況になる。これを避けるには、割り込み回数を減らし、なるべくまとめて処理する方がよい。そこで、データを受信したら XBee 内部で受信した情報を蓄積し、Arduino はタイマ割り込みを用い一定時間間隔でまとめて受信情報を読み出すようにしている。タイマ割り込みで XBee クラスの receiveXBeeData メンバ関数を呼び出してデータをすべて受け取り、そのサイズからデータを受け取ったかどうかを判断して、受け取った場合には、何かしらの受信処理を行う。

この受信処理には 2 つの処理タイミングが考えられる。一つは、割り込みでデータを受け取ったらすぐに処理する即時処理である。すぐに処理できるため即応性を確保できるが、処理に時間がかかると同一割り込み内で処理しきれず、通信バッファが溢れるなどの不具合が発生し、全体の動作に影響を及ぼす可能性がある。もう一つは、割り込み処理自体はすぐに終了し、割り込み禁止区域外で処理を行う通常処理である。処理に時間がかかった場合でも、割り込みにより定期的にデータの取得を行うことができる。下記のコードでは、これらの処理の場所を確認できる。

```

void sendmsg(String message){
    myxbee.broadcastXBeeData(message);
}

int recvret(0);
void recvmsg(void){
    String receiveStr = myxbee.receiveXBeeData();
    MsTimer2::stop();
    if(msrlen > 0){
        // 重複割り込み発生
        Serial.println("ERR:Duplicated RCV Interrupt");
    }
    msglen = receiveStr.length();
    if (msglen > 0){ // パケットを受信した
        int messagelen = receiveStr.substring(0, 3).toInt();
        int loc = 3;
        message = receiveStr.substring(loc, loc+messagelen);
        // [即時処理]変数代入など簡単な処理を記述、時間のかかる処理はアウェイ
    }
    MsTimer2::start();
    // [通常処理]比較的時間のかかる処理はここで記述することができる。
    if(msrlen > 0){
        // message;
    }
}

```

この後に、`setup` 関数と `loop` 関数を定義する。最小限の骨格だけ記せば、次のようなコードになる。

```

void setup() {
    Serial.begin(9600);
    myxbee.init(Serial);
    pinMode(LED, OUTPUT);
    MsTimer2::set(RECVINTERVAL, recvmsg); // タイマー割り込みをかける
    MsTimer2::start();
}

void loop() {
    sendmsg("送りたい値,");
    delay(1000); // データは頻繁に送信しないように
}

```

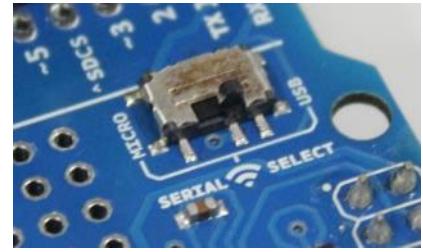
`Sendmsg`において、送りたい値の前後にコンマ”,”が配置されているが、これは API モードを利用するため、パケットの様々な情報もデータと併せて送信される。つまり、データを受ける側は様々なゴミの間に埋もれて情報が届くことになる。そのゴミの中から必要な情報を探し出すためのマーカーとしてコンマを利用することを狙っている。これについては、Node-RED の設計で説明を追加する。

5.7. Arduino へのプログラムの書き込み

Arduino と PC を接続し、を押してコンパイルと書き込みを行う。新しい XBee シールドを利用している場合は特に問題なく、書き込みと通信を行うことができるが、XBee シールドに USB MICRO の

切替えスイッチがある場合は、これを操作して USB シリアルの接続を切り替える必要がある。この場合の手順は次の通りである。

1. “SERIAL SELECT” を下 “USB” (書込モード)にする。
2. “” を押す。
3. 「正常に書き込まれた」というメッセージを確認する。
4. SERIAL SELECT を上” MICRO” (動作モード)にする。



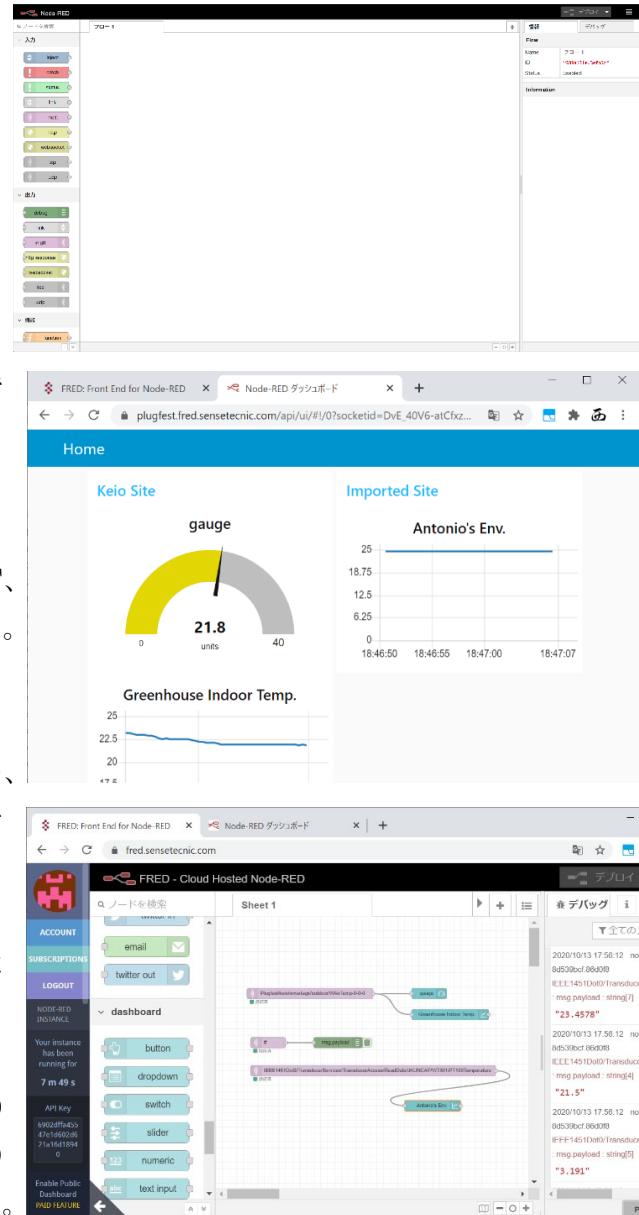
以上で書き込みが終了する。なお、シリアルモニタで Arduino と XBee 間の通信内容を確認できる。

5.8. Node-RED によるアプリケーション実装

センサのデータを可視化しアクチュエータを操作するためには、別途アプリケーションプログラムを実装し、例えばクラウド上で実行させる必要がある。ここでは、各 PC にアプリケーションプログラムを実装し、動作させることで IoT の動作内容を確認する。

ここで新たなプログラミング言語を学んでアプリケーションを設計すると IoT システム構築を実験時間内に終えることができない。そこで、より簡単にアプリケーションを設計するために Node-RED を利用する。Node-RED は「データの流れ」をフローで描画するビジュアルプログラムと呼ばれる手法で設計するため、テキストベースのプログラムからある程度解放される。また、データの変換や可視化などあらゆる処理を実現できるため、お手軽である一方で、見た目の意匠性も十分利用に耐えるアプリケーションを設計できる。Arduino が様々なライブラリを利用できるように、Node-RED もまた、3,000 を超えるプラグインが存在し、スマートスピーカ、データベース、メール、skype、LINE、Twitter、HomeKit など IoT、AI 関連のアプリケーション連携が簡単に設計できる。とにかく何でも「簡単なことならすぐにできる」ようになる設計ツールである。

さらに、Node-RED はブラウザ上で動作するため、インターネット環境さえあれば、世界のどこからでも操作できる。中身は Node.js により動作し、JavaScript で記述されている。また、JavaScript を用いて設計することもできるが、基本的には JavaScript の存在を意識する必要はない。Tinckercad のように、Node-RED を無償で利用できるサービスがあるので、これをを利用して Node-RED の操作を説明する。なお、実験 PC には Node-RED がインストールされており、フル機能が利用できる。

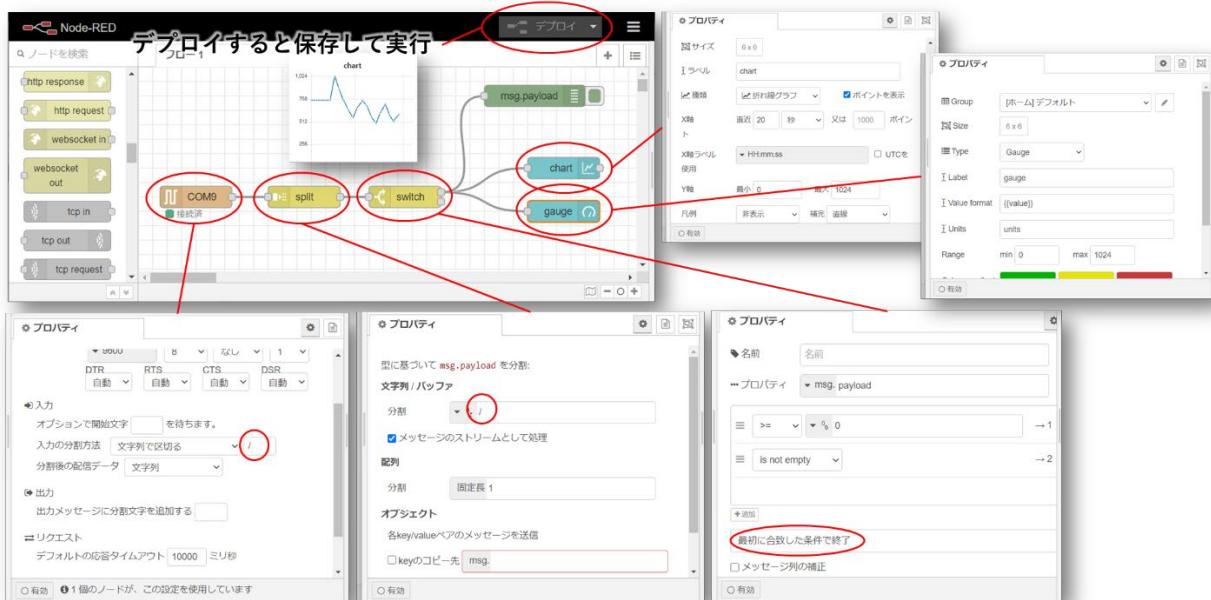


特に Node-RED のダッシュボードは簡単かつ見栄えの良いモニタ画面を構築できる。

まず、<https://users.sensetecnic.com/register> にアクセスし、FRED short を選択する。設計規模や利用時間に制限があるが、動作を学ぶ上で、この規模は障害とはならないであろう。なお、実験では、Arduino と Node-RED は PC と Arduino 間のシリアル通信を用いて相互に接続する。FRED はインターネット環境であるため、FRED と Arduino を直接接続できない。したがって、実験では FRED を利用できない。アカウントができれば Access to <https://fred.sensetecnic.com> にアクセスして FRED を利用する。Start Instance を押すと設計と動作を始めることができる。オンラインシステムであるため、他人と設計を共有することもできる。

次に、実際に受け取ったセンサの値をダッシュボードに評させる。既に説明した通り、API モードのみ無線接続の確認や無線経由の設定を行うことができるが、この API モードではコマンドを含むすべての文字が Node-RED に流れ込んでくるため、メッセージとコマンドの区別をつけることが困難になる。そこで、API モードに文字のエスケープ機能を追加した API エスケープモードを利用する。メッセージに可読文字を利用していれば、コマンドで利用される文字はエスケープされ、通常利用されない大きな文字コードを利用して送信される。この機能により、コマンドとメッセージを区別できる。なお、エスケープによりコマンドは文字化けのように見える。例えば先の例ではコンマで必要な文字や数字を挟んでいるが、このような区切り文字を頼りに、コマンドと必要な文字列を分けることができる。Arduino では、例えば s が文字配列、a が値とすると String 型の文字列変数に対して、String s = ","+String(a)+";" などとして送信し、受信側でコンマ","で囲われた部分を抽出し、数字に変換することで a を入手できる。

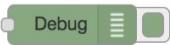
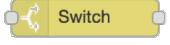
では、実際の設計例をみてみよう。Arduino からのデータを受け取るために、シリアルノードを配置する。ここでは、COM9 が指定されているが、各環境に併せて指定する必要がある。ここで、プロパティを見ると、入力文字列の分割方法が指定できる。区切り文字が / であるとすると、/ を用いて入力データを分割しておくことで不要な文字を排除しやすいようにする。なお、次の split ノードでも、区切り文字を指定して分割することができる。Switch は区切られた各文字列について条件に合致する文字列をそれぞれ該当する出力先に送り出す働きがある。ここでは、数字だけを 1 番の出力、それ以外を 2 番の出力にする。



出力に送り出すことで、数字のみを次のノードに伝える。数字だけとなつたので、これを棒グラフ(chart)とメータ(gauge)で表示する。また、同時にその内容をデバッグノードにより画面に表示する(デバッグノードに msg.payload を指定)。

ノードの動作は直観的に理解しやすいが、テキストベースのプログラムに精通している場合、流れそのものを制御するノードを用いてどのようにアルゴリズムを表現するのか戸惑うこともあるであろう。

以下、フローを直接扱うノードについて説明する。

 Inject	Inject ノードは、メッセージを投入しフローを始動させることができる。 <ul style="list-style-type: none">エディタ内でクリックすると、手動でメッセージを投入できる。一定間隔で自動的にメッセージを投入することもできる。 送出メッセージの payload および topic プロパティを設定できるが、通常は payload にメッセージと型を格納する。次の型が指定できる。 <ul style="list-style-type: none">フローまたはグローバルコンテキストのプロパティ値文字列型、数値型、Boolean型、バッファ型、オブジェクト型エポックミリ秒
 Debug	Debug ノードは、Debug サイドバーにメッセージを表示させることができる。 <ul style="list-style-type: none">メッセージを受信した時刻と受信した Debug ノードの情報が表示される。ソースノード ID をクリックすることで、ワークスペース内のどのデバッグノードなのかが判別できる。ノードのボタンで出力の有効化無効化を制御できる。ランタイムログにすべてのメッセージを出力することや、Debug ノードの下に短い(32 文字)のステータステキストを表示させることができる。
 Function	Function ノードは、JavaScript コードを直接記述して実行させることができる。 <ul style="list-style-type: none">例えば、メッセージの長さを返すならば、次のような記述を直接 function ノードに記述する。<code>var newMsg = {payload: msg.payload.length}; return newMsg;</code>
 Change	Change ノードは、メッセージプロパティ変更、コンテキストプロパティの設定ができる。各ノードには、順番に適用される複数の操作を設定でき、次のような操作が準備されている。 <ul style="list-style-type: none">値の代入 - プロパティの代入、各種型を利用でき、既存のメッセージやコンテキストプロパティも記述できる。値の置換 - メッセージプロパティの一部を検索、置換を行う。値の移動 - プロパティの移動や名称変更を行う。値の削除 - プロパティを削除し、設定に軽量クエリ変換言語 JSONata 形式が設定できる。
 Switch	Switch ノードは各メッセージを評価し、メッセージを異なるフローに振り分ける。評価するプロパティとしてメッセージプロパティおよびコンテキストプロパティを設定でき、ルールには以下の 4 種類がある。 <ul style="list-style-type: none">Value rule - 設定されたプロパティに対して評価する。Sequence rule - Split ノードで生成されるようなメッセージシーケンスを利用できる。JSONata 式 - メッセージ全体を評価し true の値を返した場合に一致したとみなす。その他は前述のルールのどれにも一致しなかった場合に一致する。 全一致に限りメッセージを送出するか、一致ルールがあった時点で評価するかを指定できる。
 template	Template ノードは、メッセージプロパティからテキストを生成する(Mustache 記法) <ul style="list-style-type: none">例えば、This is the payload: {{payload}}!
 split	Split ノードはフローを分割する
 join	join ノードはフローを結合する

5.9. MQTT によるデータの配信

様々な通信プロトコルが提案されている中で、特に IoT に向いているとされているのが、MQTT である。MQTT は一般的な通信が Peer-to-Peer、つまりサーバクライアントモデルで構築されており、通信しあう機器同士が直接メッセージを送信と受取りを行う仕組みである。より具体的には、メッセージを送信する機器は、送信する相手を唯一に規定する IP アドレスを指定して送信する必要がある。いわば、郵便システムにおける住所を用いて手紙を送るのと似ている。相手の住所がわからなければ送信できない。IoT においてもサーバクライアントモデルは普通に利用されているが、低消費電力化が重要で非力

な IoT ノードではノード数が増加するに従いこの制約が問題となる。ある IoT ノードが取得しているデータを数多くのサービスが利用したい場合、すべてのサービスノードに対して順番にコネクションを構築してメッセージを送らなければならない。これは、もとより帯域の細い末端通信網を圧迫し、スリープすることもできず、常に通信し続けることから消費電力が増大、IoT として破綻することも想定され得る。さらに、利用する相手は固定ではなく、常に変化することも想定しなければならない。変化するたび、通信相手先の一覧を更新し、適切にメッセージを送信しなければならない。

MQTT は publish-subscribe モデルであり、基本的に通信相手の数にかかわらず、送信するメッセージ数は 1 つでよく、その通信相手も MQTT ブローカで固定である。MQTT ブローカはメッセージの配信を担う中枢で、メッセージはトピックと呼ばれる共通名で管理される。ブローカへのメッセージ送信を publish と呼ぶ。メッセージを受信したい場合は、MQTT ブローカに対してトピック名を付けて subscribe を依頼すると、そのトピックに publish されたメッセージを以降受け取ることができる。subscribe するノードの数が増えても、publish されたメッセージのコピーや配信は MQTT ブローカが行うため、IoT ノードの負荷には影響しない。また、MQTT のメッセージはシンプルな構造であり、組み込み端末でも容易に実装できる。

MQTT は IP すなわちインターネットで利用される基本プロトコルを用いるため、一度 ZigBee で集めた情報を改めて MQTT としてサーバに投げる必要がある。Node-RED は MQTT クライアントとして publish や subscribe する機能が備わっている。そこで、実験用に公開している MQTT ブローカを利用して、異なるシステム間でメッセージを交換し、自分のセンサの表示に加えて、他のシステムのセンサの表示が同一画面に存在する統合サービスが構築できる。このように IoT システム間やサービスシステム間で相互に情報交換することで、より高度なサービスを構築できる。

MQTT を利用するには、上記の通り MQTT ブローカが必要である。ここでは、mosquitto と呼ばれる MQTT ブローカサーバを利用する。また、無償で利用できるオープンな MQTT ブローカサービスも利用できるので、これらのブローカを利用し、共通のトピックを用いて他のシステムと相互に情報を交換してみるとよい。

5.10. 実験後の片付け

実験終了後の片づけは極めて重要である。本実験では、実験終了後の片づけを点数化し評価に加えるため、最初に写した写真の通り、元通りに片づけること。一般に実験で利用した器具を個人で購入する意味はないが、本実験に関して言えば、IoT システム構築という機会は今後もあり得ると考えられるであろう。本実験の機材は、1 セットでおよそ 1 万円であり、実験設備としては価格も手頃なため、同じデバイスや素子を揃えれば、本実験と同じ内容を何度も試すことができる。さらに、自分なりの IoT システムを構築して利用することも可能である。

近年、このような IoT システムや組み込みシステムを手軽に設計するためのデバイスが様々販売されており、他にも Raspberry Pi と呼ばれる Linux ベースの OS が稼働し、PC としても利用できる安価なコントローラも販売されている。Raspberry Pi 上で Node-RED を実行することも、Arduino IDE を実行することも可能である。このように、IoT ノードとして必要十分な機能を持っており、Arduino よりも性能が高い。カメラやディスプレイなども搭載でき、機械学習なども小規模であれば実装でき、価格も本実験の器材とほぼ同額で、本体とメモリ、ケースなど動作に必要最低限のパーツが揃う。このような様々な IoT システムにぜひ触れていただき、技術の進歩を確認していただきたいのと、IoT システム

の設計や利用は極当たり前で、誰でも簡単にできることであり、何も特別なことではないということを知って頂きたい。そして、レゴブロックを単に組み合わせただけでは研究にも論文にもならないのと同様、本実験のような内容では研究には当たらない。身近になった IoT は道具であり、その道具を用いて何を新規性とするのかを考えなければならない。IoT でビジネスをする場合も、技術的に特別なことではなく、誰でもできることであり、何をユーザのメリットとするのかを考えなければならない。

6. レポートとディスカッション

6.1. レポートの作成について

レポートには実験終了時に指示された課題に対する回答も漏れなく記載すること。プログラムコードや動作イメージなど、再現性や正確さが評価される。記述においては次の点に注意すること。

- レポートは手書き禁止である。本実験において手書きは意味がない。IT 系で手書きを求める会社に勤めるべきではない。レポートは MS Word などを利用して書くこと。手書き文字のスキャンも認めない。
- 実験目的、実験理論、実験手順、実験方法、実験結果などは記述してはならない。実験レポートは写経ではなく、たとえテキストであってもこれを写すのは剽窃と同じである。コピペレポートが厳禁としながらテキストを写させるのは意味不明である。一言も写してはならない。
- 与えられた課題のみ回答すること。毎週異なる課題を出している。異なる週の課題を解いても、加点にはならない。さらに、その部分でミスがあると減点されるため、何も得をしない。与えられた課題についての発展的な内容を含む場合に限り評価する。
- オリジナリティを最も重視して評価する。その上で、LMS による自動剽窃チェックなどを利用し、この結果を加味する。
- 回路図は、写メか、エミュレータで入力して全画面キャプチャし添付すること。
- 実行画面なども記して、正しく動作していることを証明できるようにしなさい。
- 設計した回路（もしくはコード）もすべてレポートの Word ファイルに張り付けること。
- MS Word のファイルではなく、PDF で提出すること。
- 参考文献は必ず記載すること。Google 検索などを用いた場合でも参考文献をつけること。参考文献の言及があれば剽窃ではない。剽窃チェックで同じ内容が web で見つかった場合、剽窃とみなしだいか減点する。残念ながら、偶然にも一致してしまった場合でも減点の対象となるので、オリジナリティの確保には細心の注意を払うこと。参考文献の記述についても、「レポートのどの部分」を「どの参考図書や参考情報から引用したか」、といった参照・被参照の両方を明記すること。
- 問題が指示する内容がわからない場合は、TA もしくは担当教員まで問い合わせること。何を質問しても減点されることではなく、遠慮することも無い。態度点は実験中のみ勘案され、レポート点はレポートの内容のみ勘案される。つまり、それ以外の時間は加味されない。つまり、その間に質問すれば、何を聞いても点数には響かない。

6.2. ディスカッション

- ディスカッションが必要と思われる場合のみ Slack もしくは LMS から直接 keio.jp アカウントに連絡を送るので、該当する場合は指定されたディスカッションの時刻にディスカッションとして

指定された部屋に行くこと。なお、担当教員がいないなどでディスカッションを始めることができない場合は、実験部屋に行き、TA もしくは担当教員に連絡すること。

- ディスカッションは基本的には個別に行うが、同様の議論が必要と判断される場合、グループで行う場合もある。Slack で指示された通りに集合すること。
- レポートの記載内容についての確認を行うことが多いため、あらかじめ課題を考える際には、レポートの記述内容に対して、どうしてそのような記述をしたのかという理由を考えておくこと。
- 本実験はディスカッションにおける回答や理由の正確さよりも、より積極性や思考の深さ、そのオリジナリティを高く評価する。先輩や既に実験を終えた学生に聞けば、いくらでも正答を得ることができる。そのようなディスカッションもレポートも期待していない。単純に正答を追いかけたレポートは評価されないことに留意すること。本実験の目的は「正しい答えを導くこと」ではない。

6.3. レポート課題例

1. LED 制御

ボタンを押すと LED が 1 秒点灯し、離すと別の LED が 1 秒点灯するようにせよ。

2. リモート制御

可変抵抗とサーボモータを組み合わせて、可変抵抗でサーボを制御せよ。すなわち、リモートロボットを作成せよ。（<https://www.arduino.cc/en/Tutorial/Sweep> 参考）

3. タイマ制御

タイマ関数と Callback を用いて、ボタンを押したら 2 秒間点灯するようにせよ。

4. 正確な時間制御

毎秒 LED が 200ms 点灯するようにせよ。ただし、正確に毎秒点灯すること。sleep などを用いると、sleep 以外の処理時間が加算されるため正確ではない。

5. 足し算電卓

足し算電卓を作成せよ。ボタンは 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, +, = があればよい。計算結果は、シリアル通信を行い、Arduino IDE の知るあるモニタ上に表示させること。

6. シリアル電卓の作成

シリアルから、式を入力してもらい、シリアルで答えを返す電卓を作成せよ。受け付けるのは上記と同じ 0-9 と +, = である。

7. ビンゴ抽選機

ビンゴ抽選機を実装せよ。抽選ボタンを押すとビンゴ抽選を行い、ある数を「重複なく」シリアル経由でシリアルモニタへ出す。すべての球つまり数字が出現したら、「終了」LED を点灯させる。リセットボタンを押すと、初期状態に戻り、再び抽選できる。抽選の数字は 1 から 30 としなさい。

8. Arduino 間 LED 制御

2 個の Arduino を 1 本で接続し、片側の Arduino のボタンを押すと、もう片方の Arduino の LED が 1 秒点灯し、同様に離すと別の LED が 1 秒点灯するようにしなさい。

9. Arduino 間サーボ制御

2 個の Arduino をパラレル接続し、片側の Arduino のつまみの角度に応じて、もう片方の

Arduino のサーボが動作するようにせよ。

10. Arduino 間シリアル通信

2 個の Arduino をシリアル接続し、送信した a-z の文字をシリアルで出すようにせよ。配線電圧の HIGH、LOW を制御して情報を伝えるようにすること。

11. Arduino 間シリアル通信によるサーボ制御

2 個の Arduino をシリアル接続し、シリアル通信で片側の Arduino のつまみの角度に応じて、もう片方の Arduino のサーボが動作するようにせよ。

12. カスケードシリアルによる LED 制御

3 個の Arduino をカスケードシリアル接続し LED を制御せよ。プライマリとなる Arduino のボタンを押すと、残りすべての Arduino の LED が 1 秒点灯する。

(ヒント) カスケードなので、プライマリ以外の Arduino は親からの情報の受信と子への送信を同時にを行うことになる。なお、発展的な内容として、プライマリを定めず、どの Arduino も他の Arduino に対して指令を出すこともできる。

13. カスケードシリアルによるサーボ制御

3 個の Arduino をカスケードシリアル接続しサーボを制御せよ。いずれか一つの Arduino に接続されたつまみの角度に応じて、残りの Arduino のサーボが動作する。なお、発展的な内容として、プライマリを定めず、どの Arduino も他の Arduino に対して指令を出すこともできる。

14. カスケードシリアルによる通信

3 個の Arduino をカスケードシリアル接続し送信した a-z の文字をシリアルで出すようにせよ。配線電圧の HIGH、LOW を制御して伝えるようにすること。プライマリとなる Arduino から送信すると残りすべてに伝わるようにする 3 個の Arduino をカスケードシリアル接続し送信した a-z の文字をシリアルで出すようにせよ。配線電圧の HIGH、LOW を制御して伝えるようにすること。なお、発展的な内容として、なお、プライマリを定めず、どの Arduino も他の Arduino に対して指令を出すこともできる。

15. カスケードシリアルによるループ通信

3 個の Arduino をカスケードシリアル接続し送信した a-z の文字をシリアルで出すようにせよ。なお、マスターを規定せず、カスケード接続をループで構成して、指令を出した Arduino 以外の Arduino で文字が出るようにせよ。配線電圧の HIGH、LOW を制御して伝えるようにすること。すべての Arduino が送信・受信両方共できるようにすること。

16. シリアルバスによる LED 制御

3 個の Arduino をバスシリアル接続し LED を制御せよ。いずれか一つの Arduino のボタンを押すと、残りの Arduino の LED が 1 秒点灯する、同様に離すと、別の LED が 1 秒点灯するようにせよ。

(ヒント) バスなので、一つが送信で後の 1 つは同時に受けることになる。

17. シリアルバスによるサーボ制御

3 個の Arduino をバスシリアル接続しサーボを制御せよ。いずれか一つの Arduino に接続されたつまみの角度に応じて、残りの Arduino のサーボが動作するようにしなさい。制御する

Arduino を選択するためのボタンかスイッチも設けなさい。なお、アクティブ High に接続した配線に対して Low に繋ぐと、全体として Low になるなどする必要がある。

18. シリアルバスによる通信

3 個の Arduino をバスシリアル接続し送信した a-z の文字をシリアルで出すようにせよ。配線電圧の HIGH、LOW を制御して伝えるようにすること。どれか一つの Arduino から送信すると残りすべてに伝わるようにすること。アクティブ High に接続した配線に対して Low に繋ぐと、全体として Low になるなどするといい。

19. 自動調歩通信

2 個の Arduino 間でボーレートを合わす必要がないようにシリアル接続せよ。なお、2 本配線してよい。

20. 自動調歩・高信頼通信

2 個の Arduino を次の仕様でシリアル接続せよ。プロトコルは自由に設計してよい。

- ボーレートを合わす必要がないようにすること(2 本配線してよい)
- 通信中に配線を外すとエラー表示を出すこと

21. 1-wire 自動調歩通信

2 個の Arduino を次の仕様でシリアル接続せよ。プロトコルは自由に設計してよい。

- ボーレートを合わす必要がないようにすること(配線 1 本のみで通信すること)
- 100bps (動作限界は環境に依存する) までならばどのようなボーレートでも通信可能とすること。

22. エラー検出機能付き 1-wire 自動調歩通信

2 個の Arduino を次の仕様でシリアル接続せよ。プロトコルは自由に設計してよい。

- ボーレートを合わす必要がないようにすること(配線 1 本のみで通信すること)
- 100bps (動作限界は環境に依存する) までならばどのようなボーレートでも通信可能とすること。
- 通信中に配線を外すとエラー表示を出すこと。

7. 実験環境の初期インストール

実験機材を交換する、新たに導入する、個人で環境を構築するなどの際には、実験環境をセットアップが必要である。学生が自主的に環境を構築しさらに学ぶ機会をサポートするため、その手順について説明する。

7.1. 機材

次の機材を購入するなどして導入する。

- オンラインショップで購入可能な Arduino スターターや学習キットを導入する。基本的にどのようなキットでも構わない。実験では、Elegoo の UNO R3 Project “The Most Complete Starter Kit”を利用している。この中に Arduino 互換品が含まれている。また、各種センサ、モータ、ブレッドボード、抵抗、USB ケーブルなども含まれているはずである。各自内容物については確認すること。

スターターキットを利用すれば基本的な実験を行うことができるが、ZigBee 環境は一般的なスタータ

キットには含まれていない。したがってオンラインショップで XBee を購入することになる。必要なデバイスは次の通りである。

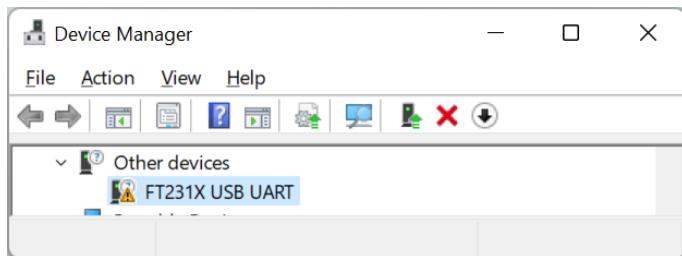
- 実験では一般的な ZigBee モジュールとして XBee S2C を利用している。なお、Arduino と PC で通信する場合、それぞれ 1 個つまり、合計で 2 個は最低必要となる。
- XBee シールドとして Arduino 用 XBee シールド(スイッチサイエンス製)を利用している。
- XBee を PC に接続するために XBee USB アダプタ(スイッチサイエンス製 rev.2)を利用している。

7.2. XBee の設定

Arduino 関連は直接設計プログラムを書き込むため、特に初期設定は必要ないが、XBee はそれ自体にマイクロコントローラを搭載しており、各種設定が行えるようになっている。通信を行うには、通信相手を特定するため、それぞれの物理 ID などを設定する必要があり、通信グループや、プロトコルの種別などを最低でも設定する必要がある。設定手順は以下の通りである。

まず、USB に XBee を搭載した XBee USB アダプタを接続する。デバイスマネージャが、次のように表示される場合認識ができておらず利用することができない。

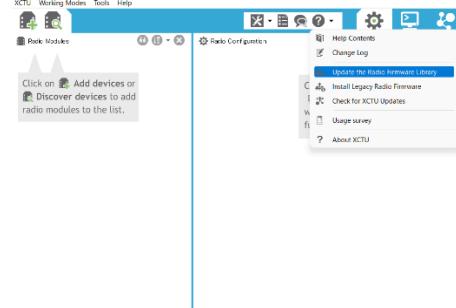
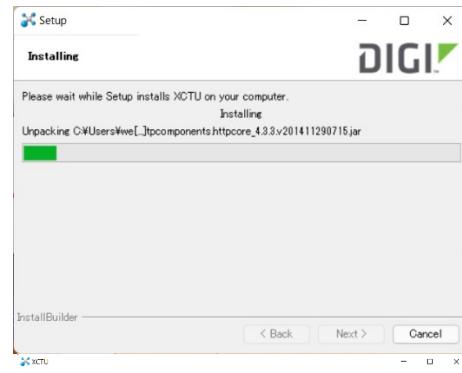
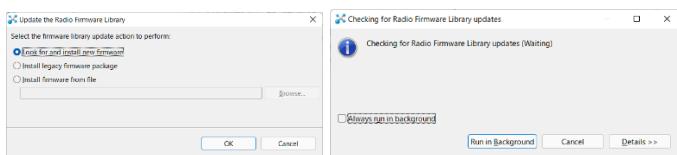
<https://ftdichip.com/drivers/vcp-drivers/>において Windows 用ドライバ(CDM212364_Setup)をダウンロードし、インストールする。



XBee は Digi 社の製品であり、Digi より提供されている XCTU をを利用して設定する。

Digi のサイトから XCTU をダウンロードしてインストールする。

XCTU を起動する。初めて起動する際は、XBee のファームウェアの最新版をダウンロードする必要がある。ダウンロードしたファームウェアを XBee に焼き込むことで最新の状態に保つことができる。”Update the Radio Firmware Library”を選択し、ダウンロードする。



Xbee を Xbee USB アダプタに取付け(向きに注意すること)、USB に接続した後、虫眼鏡アイコンを押し、接続されている USB シリアルポート(COM 数字)を選択して Next を押す。シリアルに関する様々な設定を行い(何も変更していないければデフォルトで構わない)、Finish を押す。

XBeeを見つけるとアイコンで表示される。アイコンの C の赤い文字はコーディネータとして設定されていることを表す。コーディネータは、構築するシステム内に1つだけ存在し、複数存在してはならない。実験環境では4つのXBeeを利用し、1つをコーディネータとしてPCに接続する。RF802はZigBeeの物理層の規格 IEEE 802.15.4 RFであることを表す。Digi は同じ形状で WiFi モジュールなども提供しており、モジュールの種別を区別することができる。

設定したいモジュールを選択して、Add selected devices を押す。

調べたい XBee をクリックして、内部設定情報を見る。右の図は購入したときの XBee の設定であり、このモードは利用しない。Firmware を書き換えてメニューを変更する。Function set に記載されているのが Firmware の構成であり、802.15.4 TH がデフォルトである。これを、ZIGBEE TH Reg に修正する。

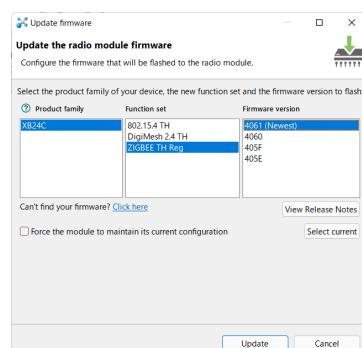
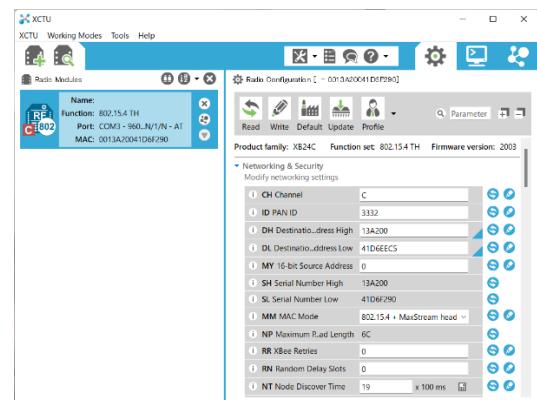
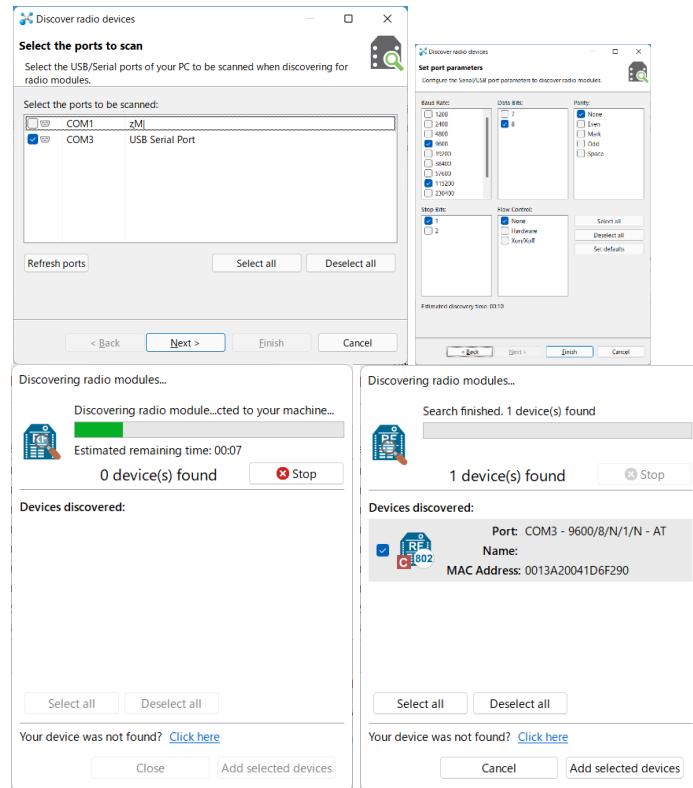
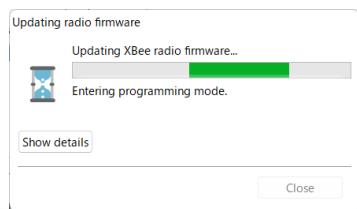
なお、Firmware には次の種類がある。

802.15.4 TH : Point to point(1対1), Point to Multi Point として利用

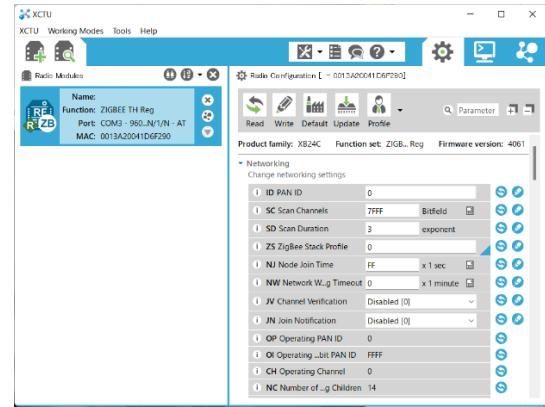
DigiMesh 2.4 TH : DigiMesh プロトコルを利用

ZIGBEE TH Reg : ZigBee プロトコルを利用

次に、ファームウェアを更新する。Update を押して、XB24C の ZIGBEE TH Reg の最新バージョンを選択し、Update を押すことで更新できる。



更新に成功すると、ZIGBEE TH Reg の設定内容が見えるようになる。



次の表にしたがって、設定内容を確認する。下記は、Firmware Version 4061、Hardware Version 2E46に基づいている。デフォルトから設定値が異なる、一般的ではないなど特に注意するべき設定について赤で示しており、XCTU では青色の右下三角マークが現れる。変更を施し、Write する際に更新されるパラメータは、XCTUにおいて緑色の右下三角マークが現れる。修正の際の参考にするとよい。ID (PAN ID)は個人で利用する場合はどのようにつけてもよいが、実験環境では、242201から24220Fまでの番号(24・実験第2・テーマ番号2・連番)が個別に振られている。なお PAN ID は 16 衔(64bit)まで指定できる。CE は一つだけ Coordinator つまり Enabled に設定し、それ以外はすべて Router つまり Disabled に設定する。なお、End Device としては設定できず、必ず Router および Coordinator とセットとなる。

AP (API Mode)は 2 (API enabled with escaping)とする。Addressing の SL はメモしておくと良いだろう。なお、この番号は XBee モジュールの裏面にも記載されている。

Networking	Addressing	ZigBee Addressing	Serial Interfacing	Sleep Modes
<ul style="list-style-type: none"> ■ ID: 242201-(順番) ■ SC: 7FFF ■ SD: 3 ■ ZS: 0 ■ NJ: FF ■ NW: 0 ■ JV: 1 ■ JN: 0 ■ OP: 3320 ■ OI: A6D ■ CH: 14 ■ NC: 14 ■ CE: <ul style="list-style-type: none"> ■ Router: 0 ■ Coordinator: 1 ■ DO: 0 ■ DC: 0 	<ul style="list-style-type: none"> ■ SH: 13A200 ■ SL: 41637A8A (P2P通信が必要) ■ MY: C262 ■ MP: FFFE ■ DH: 0 ■ DL: 0 ■ NI: ■ NH: 1E ■ BH: 0 ■ AR: FF ■ DD: A0000 ■ NT: 3C ■ NO: 0 ■ NP: FF ■ CR: 3 	<ul style="list-style-type: none"> ■ SE: E8 ■ DE: E8 ■ CI: 11 ■ TO: 0 <p>RF Interfacing</p> <ul style="list-style-type: none"> ■ PL: 4 ■ PM: 1 ■ PP: 8 <p>Security</p> <ul style="list-style-type: none"> ■ EE: 0 ■ EO: 0 ■ KY: 00 ■ NK: 00 	<ul style="list-style-type: none"> ■ BD: 3 ■ NB: 0 ■ SB: 0 ■ RO: 3 ■ D6: 0 ■ D7: 1 ■ AP: 2 ■ Node-REDとCoordinatorをつなげる際の便宜を考え、0にする場合もあり ■ AO: 0 <p>AT Command Options</p> <ul style="list-style-type: none"> ■ CT: 64 ■ GT: 3E8 ■ CC: 2B 	<ul style="list-style-type: none"> ■ SP: 20 ■ SN: 1 ■ SM: 0 ■ ST: 1388 ■ SO: 0 ■ WH: 0 ■ PO: 0 <p>I/O Setting I/O Sampling</p>

設定が終わったら、Write を押して書き込む。以上で XBee の設定は終了である。実験に利用するすべての XBee を上記のように設定すること。また、次の点に注意すること

- Arduinoに乗ったままでは設定できない。必ず USB アダプタに刺してから設定すること。
- 抜き差しは慎重に。特に抜くときは足を曲げないように。指すときは逆差ししないように。
- XCTU で見えない場合は次の手順で初期化すると解決する場合がある。その前に、必ず接続や電源、シリアルの設定などに誤りがないか今一度確認しておくこと。

解法 1 : XCTU のシリアルコンソールを開いて reset を押し、+++と+を連打する。シリアルコンソールの画面に、OK と返事が戻ったら、すかさず、AT FS FORMAT CONFIRM と入力して

XBee 内部のファイルシステムを初期化する。

解法2：ファームウェアを更新する。

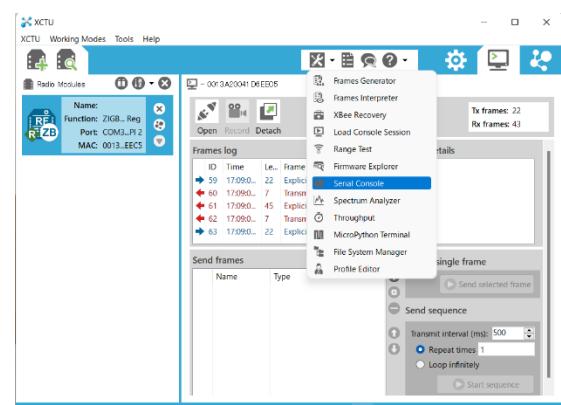
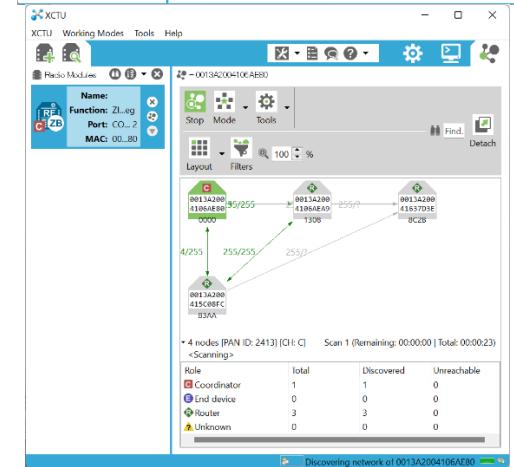
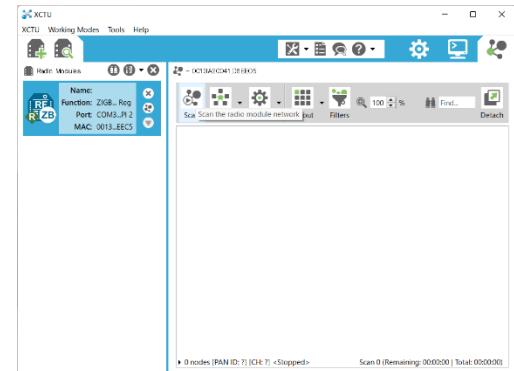
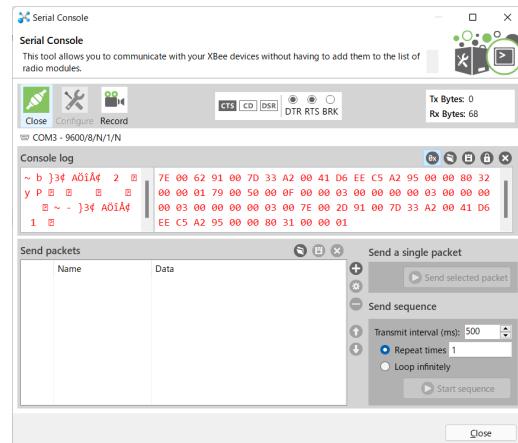
7.3. XBee の検索と ZigBee ネットワークの確認

既に設定が終わった XBee モジュールがあるならば、すべて電源に接続する。Coordinator は USB アダプタに刺して PC に、その他は XBee シールドに刺して Arduino に接続し USB で電源を供給する。

Scan を押して、モジュールを検索する。

実験では 4 個の XBee が接続されているため、4 つのモジュールが見える。図のように相互に認識され、ZigBee ネットワークが構築されていることがわかる。また、認識している通信相手や、利用する通信経路などの情報が表示される。C はコーディネータで 1 つだけ存在し、R がルータでこれらが相互に接続されているはずである。

Tool の Serial Console を選択し、Open を押して Close にすると、受け取ったパケットの情報が表示される。必ずしも通信した情報だけでなく、ZigBee ネットワークの維持や管理に必要な情報も表示される点に注意すること。



以上で XBee の設定と動作の確認は終了である。

7.4. Arduino IDE・Node-RED・Mosquitto のインストール

7.4.1. Arduino IDE

- Arduino のホームページ(<https://www.arduino.cc/en/software>)からダウンロードする。Windowsならば Microsoft Store で Arduino IDE を検索してインストールすることもできる。

7.4.2. Node-RED

- Node-RED Users Group Japan が様々な情報を公開している。
- Node-RED は JavaScript で記述されているため、その実行には Node.js が必要となる。Node.js は Node.js 公式ホームページからインストールする。
- Node.js のパッケージマネージャ npm を使って、”npm install -g --unsafe-perm node-red”として Node-RED をインストールする。インストールが終われば、コマンドラインで node-red と直接実行ファイルを起動する。
- Node-RED は web サーバとして動作し、node-red 起動時に出力される URL、もしくは localhost:1880 にアクセスする。
- Node-RED のアドオンをインストールするため、node-red のブラウザ画面上、右上三本線のアイコンをクリックし、パレットの管理、ノードを追加を選択する。
- シリアルポートインターフェースをインストールするため、ノードを検索の中に、serial と入力し、node-red-node-resialport のノードを追加する。
- ノードを検索の中に、dashboard と入力し、node-red-dashboard のノードを追加する。

7.4.3. Mosquitto

- Windows 用 Mosquitto は Mosquitto のページ(<https://mosquitto.org/download/>)からダウンロードする。今回のように、特にセキュリティなど設定しないのであれば、オープンで無償の MQTT ブローカサービスを利用するとよい。
- Mosquitto が正しく動作しているかどうかを確認するには、次のように実際にメッセージを送受信するとよい。
- まず、subscriber を起動する。”mosquitto_sub -h localhost -t test”として、localhost つまり、自分のマシンの上で動作している mosquitto にアクセスし、トピック名を test として subscribe する。
- 次に publish する。別のウィンドウで” mosquitto_pub -h localhost -t test -m "test message" とし、subscribe と同じトピック名 test を指定して、-m に続くメッセージを送信する。
- subscriber の画面に test message と表示されれば動作が確認できる。

7.5. 各種センサの使い方

キット内の LED やサーボモータなどは一般的な記述で動作する。しかしながら、動作に工夫の必要なデバイスも含まれているため、以下にサンプルを示す。

7.5.1. ウルトラソニックセンサ

別途配布される SR04 ドライバを利用する。

```
#include "SR04.h"
#define TRIG_PIN 12
#define ECHO_PIN 11
SR04 sr04 = SR04(ECHO_PIN,TRIG_PIN);
long a;
void setup() {
```

```

Serial.begin(9600);
delay(1000);
}
void loop0 {
  a=sr04.Distance();
  Serial.print(a);
  Serial.println("cm");
  delay(1000);
}

```

7.5.2. 薄膜キーボード

配布される Keypad ライブライアリを利用する。

```

#include <Keypad.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
//define the symbols on the buttons of the keypads
char hexaKeys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {9, 8, 7, 6}; //connect to the row pinouts of the keypad
byte colPins[COLS] = {5, 4, 3, 2}; //connect to the column pinouts of the keypad

//initialize an instance of class NewKeypad
Keypad customKeypad = Keypad( makeKeymap(hexaKeys), rowPins, colPins, ROWS, COLS);
void setup0{
  Serial.begin(9600);
}
void loop0{
  char customKey = customKeypad.getKey();

  if (customKey){
    Serial.println(customKey);
  }
}

```

7.5.3. デジタル温湿度センサ

配布される DHT ライブライアリを利用する。

```

#include <dht_nonblocking.h>
#define DHT_SENSOR_TYPE DHT_TYPE_11
static const int DHT_SENSOR_PIN = 2;
DHT_nonblocking dht_sensor( DHT_SENSOR_PIN, DHT_SENSOR_TYPE );
void setup() {
  Serial.begin( 9600 );
}

static bool measure_environment( float *temperature, float *humidity ) {
  static unsigned long measurement_timestamp = millis();
  if( millis() - measurement_timestamp > 3000ul ) {
    if( dht_sensor.measure( temperature, humidity ) == true ) {
      measurement_timestamp = millis();
      return( true );
    }
  }
  return( false );
}

void loop() {
  float temperature;
  float humidity;
  if( measure_environment( &temperature, &humidity ) == true ) {
    Serial.print( "T = " );
    Serial.print( temperature, 1 );
  }
}

```

```

    Serial.print( " deg. C, H = " );
    Serial.print( humidity, 1 );
    Serial.println( "%" );
}
}

```

7.5.4. アナログジョイスティック

アナログポテンショメータが XY とスイッチの 3 軸に配置されている。

```

const int SW_pin = 2; // digital pin connected to switch output
const int X_pin = 0; // analog pin connected to X output
const int Y_pin = 1; // analog pin connected to Y output
void setup() {
    pinMode(SW_pin, INPUT);
    digitalWrite(SW_pin, HIGH);
    Serial.begin(9600);
}
void loop() {
    Serial.print("Switch: ");
    Serial.print(digitalRead(SW_pin));
    Serial.print("\n");
    Serial.print("X-axis: ");
    Serial.print(analogRead(X_pin));
    Serial.print("\n");
    Serial.print("Y-axis: ");
    Serial.println(analogRead(Y_pin));
    Serial.print("\n\n");
    delay(500);
}

```

7.5.5. IR レシーバ(赤外線リモコン受信機)

配布される IRremote ライブライアリを利用する。

```

#include "IRremote.h"
int receiver = 11; // Signal Pin of IR receiver to Arduino Digital Pin 11
IRrecv irrecv(receiver); // create instance of 'irrecv'
decode_results results; // create instance of 'decode_results'
void translateIR() // takes action based on IR code received
{
    switch(results.value) {
        case 0xFFA25D: Serial.println("POWER"); break;
        case 0FFE21D: Serial.println("FUNC/STOP"); break;
        case 0xFF629D: Serial.println("VOL+"); break;
        case 0xFF22DD: Serial.println("FAST BACK"); break;
        case 0xFF02FD: Serial.println("PAUSE"); break;
        case 0FFC23D: Serial.println("FAST FORWARD"); break;
        case 0FFE01F: Serial.println("DOWN"); break;
        case 0FFA857: Serial.println("VOL-"); break;
        case 0FF906F: Serial.println("UP"); break;
        case 0FF9867: Serial.println("EQ"); break;
        case 0ffb04F: Serial.println("ST/REPT"); break;
        case 0FF6897: Serial.println("0"); break;
        case 0FF30CF: Serial.println("1"); break;
        case 0FF18E7: Serial.println("2"); break;
        case 0FF7A85: Serial.println("3"); break;
        case 0FF10EF: Serial.println("4"); break;
        case 0FF38C7: Serial.println("5"); break;
        case 0FF5AA5: Serial.println("6"); break;
        case 0FF42BD: Serial.println("7"); break;
        case 0FF4AB5: Serial.println("8"); break;
        case 0FF52AD: Serial.println("9"); break;
        case 0xFFFFFFFF: Serial.println(" REPEAT"); break;
    default:
        Serial.println(" other button ");
    }
    delay(500); // Do not get immediate repeat
}

```

```

}
void setup0 {
  Serial.begin(9600);
  Serial.println("IR Receiver Button Decode");
  irrecv.enableIRIn(); // Start the receiver
}
void loop0 {
  if (irrecv.decode(&results)) {
    translateIR0;
    irrecv.resume(); // receive the next value
  }
}

```

7.5.6. LED ドットマトリックス

配布される MAX7219 ライブラリを利用する。ここでは 3 つの表示モードの例を示している。

`rows` : `setRow` メンバ関数を利用して行で指定する。

`columns()` : `setColumn` メンバ関数を利用して列で指定する。

`single()` : マトリックスの 1 つの LED を直接指定する。

```

#include "LedControl.h"
LedControl lc=LedControl(12,10,11,1);
unsigned long delaytime1=500;
unsigned long delaytime2=50;
void setup0 {
  lc.shutdown(0,false);
  /* Set the brightness to a medium values */
  lc.setIntensity(0,8);
  /* and clear the display */
  lc.clearDisplay(0);
}
void rows0 {
  for(int row=0;row<8;row++) {
    delay(delaytime2);
    lc.setRow(0,row,B10100000);
    delay(delaytime2);
    lc.setRow(0,row,(byte)0);
    for(int i=0;i<row;i++) {
      delay(delaytime2);
      lc.setRow(0,row,B10100000);
      delay(delaytime2);
      lc.setRow(0,row,(byte)0);
    }
  }
}
void columns0 {
  for(int col=0;col<8;col++) {
    delay(delaytime2);
    lc.setColumn(0,col,B10100000);
    delay(delaytime2);
    lc.setColumn(0,col,(byte)0);
    for(int i=0;i<col;i++) {
      delay(delaytime2);
      lc.setColumn(0,col,B10100000);
      delay(delaytime2);
      lc.setColumn(0,col,(byte)0);
    }
  }
}
void single0 {
  for(int row=0;row<8;row++) {
    for(int col=0;col<8;col++) {
      delay(delaytime2);
      lc.setLed(0,row,col,true);
    }
  }
}

```

```

delay(delaytime2);
for(int i=0;i<col;i++) {
    lc.setLed(0,row,col,false);
    delay(delaytime2);
    lc.setLed(0,row,col,true);
    delay(delaytime2);
}
}
}
}

void loop0 {
rows0;
columns0;
single0;
}

```

7.5.7. ジャイロセンサ(MPU-6050)

別途配布される MPU-6050 ライブライアリを利用してすることで、さらに多くの機能を利用できるが、ここでは標準で備わっている I2C インタフェースライブライアリを利用する例をしめす。

```

#include<Wire.h>
const int MPU_addr=0x68; // I2C address of the MPU-6050
int16_t AcX,AcY,AcZ,Tmp,GyX,GyY,GyZ;
void setup0{
    Wire.begin();
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x6B); // PWR_MGMT_1 register
    Wire.write(0); // set to zero (wakes up the MPU-6050)
    Wire.endTransmission(true);
    Serial.begin(9600);
}
void loop0{
    Wire.beginTransmission(MPU_addr);
    Wire.write(0x3B); // starting with register 0x3B (ACCEL_XOUT_H)
    Wire.endTransmission(false);
    Wire.requestFrom(MPU_addr,14,true); // request a total of 14 registers
    AcX=Wire.read()<<8|Wire.read(); // 0x3B (ACCEL_XOUT_H) & 0x3C (ACCEL_XOUT_L)
    AcY=Wire.read()<<8|Wire.read(); // 0x3D (ACCEL_YOUT_H) & 0x3E (ACCEL_YOUT_L)
    AcZ=Wire.read()<<8|Wire.read(); // 0x3F (ACCEL_ZOUT_H) & 0x40 (ACCEL_ZOUT_L)
    Tmp=Wire.read()<<8|Wire.read(); // 0x41 (TEMP_OUT_H) & 0x42 (TEMP_OUT_L)
    GyX=Wire.read()<<8|Wire.read(); // 0x43 (GYRO_XOUT_H) & 0x44 (GYRO_XOUT_L)
    GyY=Wire.read()<<8|Wire.read(); // 0x45 (GYRO_YOUT_H) & 0x46 (GYRO_YOUT_L)
    GyZ=Wire.read()<<8|Wire.read(); // 0x47 (GYRO_ZOUT_H) & 0x48 (GYRO_ZOUT_L)
    Serial.print("AcX = "); Serial.print(AcX);
    Serial.print(" | AcY = "); Serial.print(AcY);
    Serial.print(" | AcZ = "); Serial.print(AcZ);
    Serial.print(" | Tmp = "); Serial.print(Tmp/340.00+36.53); // in degrees C from datasheet
    Serial.print(" | GyX = "); Serial.print(GyX);
    Serial.print(" | GyY = "); Serial.print(GyY);
    Serial.print(" | GyZ = "); Serial.println(GyZ);
    delay(333);
}

```

7.5.8. 人感センサ(人体赤外線感応モジュール)

デジタル信号入力で検出できる。

```

int ledPin = 13; // LED on Pin 13 of Arduino
int pirPin = 7; // Input for HC-S501
int pirValue; // Place to store read PIR Value
void setup0{
    pinMode(ledPin, OUTPUT);
    pinMode(pirPin, INPUT);
    digitalWrite(ledPin, LOW);
}

```

```

void loop0{
    pirValue = digitalRead(pirPin);
    digitalWrite(ledPin, pirValue);
}

```

7.5.9. 水位センサ

水位をアナログ値として検出できる。

```

int adc_id = 0;
int HistoryValue = 0;
char printBuffer[128];
void setup0{
    Serial.begin(9600);
}
void loop0{
    int value = analogRead(adc_id); // get adc value
    if((HistoryValue>=value) && ((HistoryValue - value) > 10)) || ((HistoryValue<value) && ((value - HistoryValue) > 10))) {
        sprintf(printBuffer,"ADC%d level is %d\n",adc_id, value);
        Serial.print(printBuffer);
        HistoryValue = value;
    }
}

```

7.5.10. リアルタイムクロックモジュール

別途配布される DS3231 ライブライアリを利用する。

```

#include <Wire.h>
#include <DS3231.h>
DS3231 clock;
RTCDateTime dt;
void setup0{
    Serial.begin(9600);
    Serial.println("Initialize DS3231");
    clock.begin();
    clock.setDateTime(__DATE__, __TIME__);
}
void loop0{
    dt = clock.getDateTime();
    Serial.print("Raw data: ");
    Serial.print(dt.year); Serial.print("-");
    Serial.print(dt.month); Serial.print("-");
    Serial.print(dt.day); Serial.print(" ");
    Serial.print(dt.hour); Serial.print(":");
    Serial.print(dt.minute); Serial.print(":");
    Serial.print(dt.second); Serial.println("");
    delay(1000);
}

```

7.5.11. 音響センサ

デジタル信号出力とアナログ信号出力がある。

- デジタル信号出力

```

int sensorPin = A0; // select the input pin for the potentiometer
int ledPin = 13; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor
void setup0{
    pinMode(ledPin,OUTPUT);
    Serial.begin(9600);
}
void loop0{
    sensorValue = analogRead(sensorPin);
    digitalWrite(ledPin, HIGH);
    delay(sensorValue);
    digitalWrite(ledPin, LOW);
}

```

```

delay(sensorValue);
Serial.println(sensorValue, DEC);
}

```

- アナログ信号出力

```

int Led=13;//define LED port
int buttonpin=3; //define switch port
int val;//define digital variable val
void setup() {
  pinMode(Led,OUTPUT);//define LED as a output port
  pinMode(buttonpin,INPUT);//define switch as a output port
}
void loop() {
  val=digitalRead(buttonpin);//read the value of the digital interface 3 assigned to val
  if(val==HIGH) {
    digitalWrite(Led,HIGH);
  } else {
    digitalWrite(Led,LOW);
  }
}

```

7.5.12. RFID モジュール

別途配布される rfid ライブライアリを利用する。

```

#include <SPI.h>
#include <MFRC522.h>
#define RST_PIN 9 // Configurable, see typical pin layout above
#define SS_PIN 10 // Configurable, see typical pin layout above
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance
#define NEW_UID {0xDE, 0xAD, 0xBE, 0xEF}
MFRC522::MIFARE_Key key;
void setup() {
  Serial.begin(9600);
  while (!Serial);
  SPI.begin();
  mfrc522.PCD_Init();
  Serial.println(F("Warning: this example overwrites the UID of your UID changeable card, use with care!"));
  for (byte i = 0; i < 6; i++) {
    key.keyByte[i] = 0xFF;
  }
}
void loop() {
  if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }
  Serial.print(F("Card UID:"));
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    Serial.print(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " ");
    Serial.print(mfrc522.uid.uidByte[i], HEX);
  }
  Serial.println();
  byte newUid[] = NEW_UID;
  if ( mfrc522.MIFARE_SetUid(newUid, (byte)4, true) ) {
    Serial.println(F("Wrote new UID to card."));
  }
  mfrc522.PICC_HaltA();
  if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    return;
  }
  Serial.println(F("New UID and contents:"));
  mfrc522.PICC_DumpToSerial(&(mfrc522.uid));
  delay(2000);
}

```

7.5.13. LCD ディスプレイ

別途配布される LiquidCrystal ライブラリを利用する。

```
/*
 * LCD RS pin to digital pin 7
 * LCD Enable pin to digital pin 8
 * LCD D4 pin to digital pin 9
 * LCD D5 pin to digital pin 10
 * LCD D6 pin to digital pin 11
 * LCD D7 pin to digital pin 12
 * LCD R/W pin to ground
 * LCD VSS pin to ground
 * LCD VCC pin to 5V
 * 10K resistor:
 *   ends to +5V and ground
 *   wiper to LCD VO pin (pin 3)
 */
#include <LiquidCrystal.h>
LiquidCrystal lcd(7, 8, 9, 10, 11, 12);
void setup() {
    lcd.begin(16, 2);
    lcd.print("Hello, World!");
}
void loop() {
    lcd.setCursor(0, 1);
    lcd.print(millis() / 1000);
}
```

7.5.14. 溫度センサ

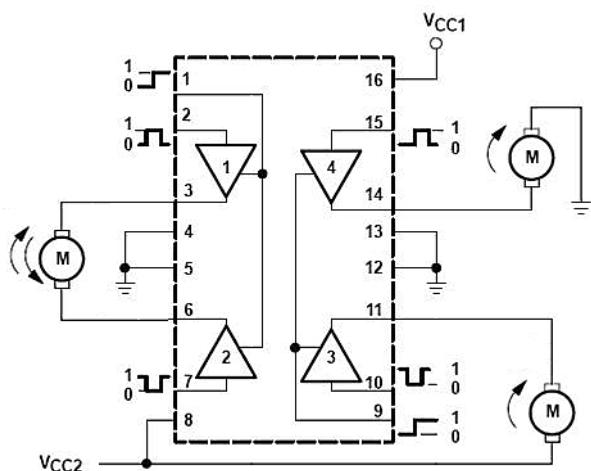
次の変換式を用いてアナログ値から温度を得る。

```
int tempReading = analogRead(tempPin);
double tempK = log(10000.0 * ((1024.0 / tempReading - 1)));
tempK = 1 / (0.001129148 + (0.000234125 + (0.0000000876741 * tempK * tempK)) * tempK );
float tempC = tempK - 273.15;           // Convert Kelvin to Celcius
float tempF = (tempC * 9.0) / 5.0 + 32.0; // Convert Celcius to Fahrenheit
```

7.5.15. 74HC595 シフトレジスタによる LED 点灯

```
int tDelay = 100;
int latchPin = 11; // (11) ST_CP [RCK] on 74HC595
int clockPin = 9; // (9) SH_CP [SCK] on 74HC595
int dataPin = 12; // (12) DS [S1] on 74HC595
byte leds = 0;
void updateShiftRegister0 {
    digitalWrite(latchPin, LOW);
    shiftOut(dataPin, clockPin, LSBFIRST, leds);
    digitalWrite(latchPin, HIGH);
}
void setup() {
    pinMode(latchPin, OUTPUT);
    pinMode(dataPin, OUTPUT);
    pinMode(clockPin, OUTPUT);
}
void loop() {
    leds = 0;
    updateShiftRegister0();
    delay(tDelay);
    for (int i = 0; i < 8; i++) {
        bitSet(leds, i);
        updateShiftRegister0();
        delay(tDelay);
    }
}
```

7.5.16. L293D を利用した DC モータ制御



```
#define ENABLE 5 // L293D(1)
#define DIRA 3 // L293D(2)
#define DIRB 4 // L293D(7)
void setup0 {
    pinMode(ENABLE,OUTPUT);
    pinMode(DIRA,OUTPUT);
    pinMode(DIRB,OUTPUT);
    Serial.begin(9600);
}
void loop0 {
    Serial.println("One way, then reverse");
    digitalWrite(ENABLE,HIGH); // enable on
    digitalWrite(DIRA,HIGH); //one way
    digitalWrite(DIRB,LOW);
    delay(500);
    digitalWrite(DIRA,LOW); //reverse
    digitalWrite(DIRB,HIGH);
    delay(500);
    digitalWrite(ENABLE,LOW); // disable
    delay(2000);
    Serial.println("fast Slow example");
    digitalWrite(ENABLE,HIGH); //enable on
    digitalWrite(DIRA,HIGH); //one way
    digitalWrite(DIRB,LOW);
    delay(3000);
    digitalWrite(ENABLE,LOW); //slow stop
    delay(1000);
    digitalWrite(ENABLE,HIGH); //enable on
    digitalWrite(DIRA,LOW); //one way
    digitalWrite(DIRB,HIGH);
    delay(3000);
    digitalWrite(DIRA,LOW); //fast stop
    delay(2000);
    Serial.println("PWM full then slow");
    analogWrite(ENABLE,255); //enable on
    digitalWrite(DIRA,HIGH); //one way
    digitalWrite(DIRB,LOW);
    delay(2000);
    analogWrite(ENABLE,180); //half speed
    delay(2000);
    analogWrite(ENABLE,128); //half speed
    delay(2000);
    analogWrite(ENABLE,50); //half speed
    delay(2000);
```

```
digitalWrite(ENABLE,LOW); //all done  
delay(10000);  
}
```

7.5.17. ステッピングモータ制御

配布されるステッパー ドライバライブラリとキットのドライバを利用する。

```
#include <Stepper.h>  
const int stepsPerRevolution = 2048; // change this to fit the number of steps per revolution  
const int rolePerMinute = 15;      // Adjustable range of 28BYJ-48 stepper is 0~17 rpm  
Stepper myStepper(stepsPerRevolution, 8, 10, 9, 11);  
void setup() {  
    myStepper.setSpeed(rolePerMinute);  
    Serial.begin(9600);  
}  
void loop() {  
    Serial.println("clockwise");  
    myStepper.step(stepsPerRevolution);  
    delay(500);  
    Serial.println("counterclockwise");  
    myStepper.step(-stepsPerRevolution);  
    delay(500);  
}
```

S D実験 IoT テキスト

Ver. SDED-F2-2022-04-01