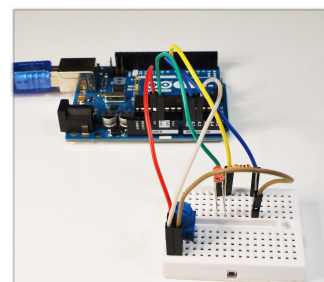
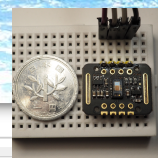
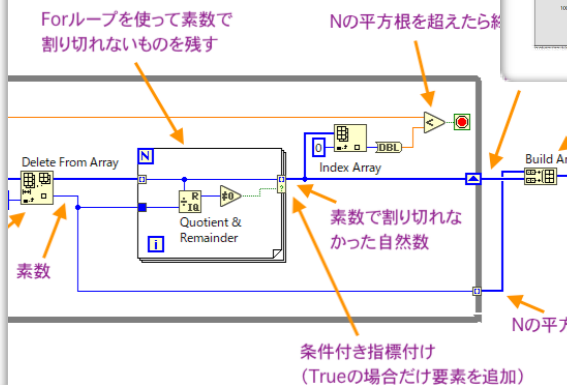
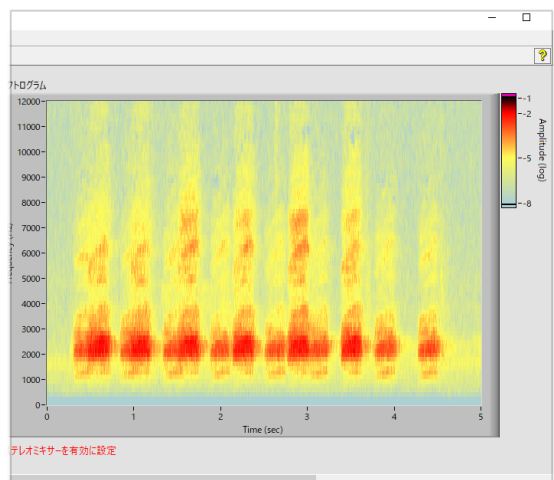


# 趣味で使う無料のLabVIEW



LabVIEW Community Editionで  
プログラミングを楽しもう



著作：日本LabVIEWユーザー会有志

Copyright (c) 2020 Japan LabVIEW Users Group, Volunteer members.



This work is licensed under a Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License.

この書籍に添付されるプログラムはMITライセンスで提供されます。

All programs attached to this book is provided under the MIT license.

Copyright (c) 2020 Japan LabVIEW Users Group, Volunteer members.

<http://opensource.org/licenses/mit-license.php>



# 序文

LabVIEWの生みの親であるJeff Kodoskyから、LabVIEW Community Editionへの想いと、読者の皆様に向けて以下のようなメッセージを頂きました。

We created the LabVIEW Community editions so engineers could use the software for free — to pursue their hobbies and personal projects, experiment with programming ideas and create and share IP with their peers. I am thrilled to see the Japan LabVIEW User Group create this e-book that contains everything a budding hobbyist would need to start programming with LabVIEW. The quality of this work shows we have a very passionate community of LabVIEW developers who want to share their love of LabVIEW with everyone. I personally want to thank the Japan LabVIEW User Group for their enthusiasm, dedication, and hard work. I look forward to seeing the many fantastic projects everyone will create with the LabVIEW Community edition after reading and learning with this free e-book.

Jeff Kodosky  
Inventor of LabVIEW  
Cofounder and Business and Technology Fellow

## 日本語訳

エンジニアが趣味や個人的なプロジェクトを追求したり、プログラミングのアイデアを試したり、IPを作成して仲間と共有したりするために、無料でソフトウェアを使用できるようにLabVIEW Community Editionを作成しました。日本LabVIEWユーザー会がLabVIEWを使ってプログラミングを始めるために必要な情報が全て詰まった電子書籍を作成してくれたことに感謝しています。この作品の質の高さは、LabVIEWへの愛をみんなと共有したいと思っているLabVIEW開発者の情熱的なコミュニティがあることを示しています。私は個人として日本LabVIEWユーザー会の熱意、献身、そして努力に感謝しています。この無料の電子書籍を読んで学んだ後に、LabVIEW Community Editionを使って皆さんが作成する多くの素晴らしいプロジェクトを見るのを楽しみにしています。

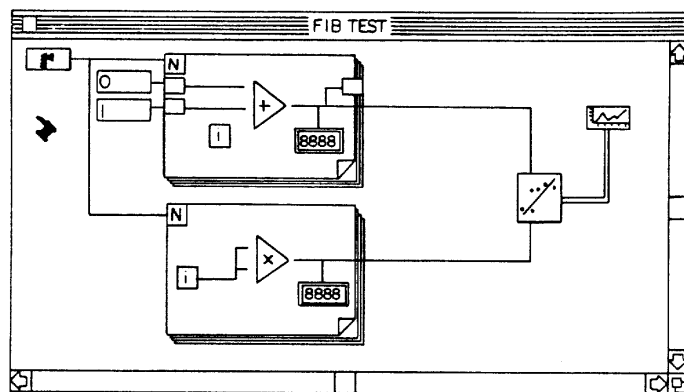
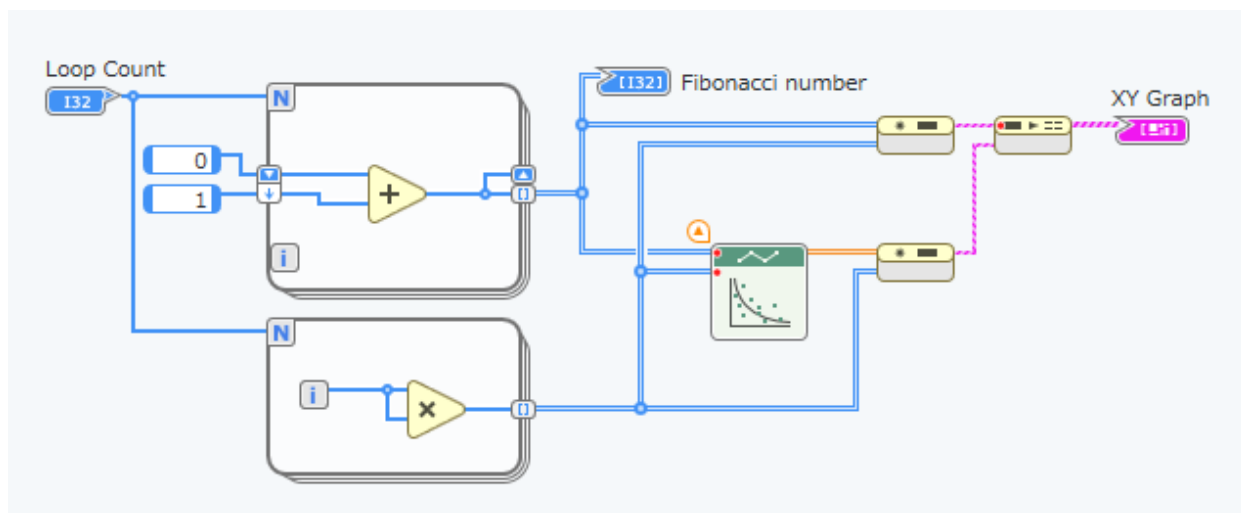


FIG. -57



# はじめに

皆さんは「プログラミング」と聞いて何をイメージしますか？皆さんが良く使うパソコンやスマートフォンは、実はプログラミングのおかげで動いています。プログラミングはいたるところで使われていて、テレビも冷蔵庫も、家電製品でプログラミングを利用していないものはほとんどありません。実感はしにくいのですが、それほどプログラミングは皆さんの身近なものになっています。

さて2020年から学校でもプログラミングを勉強することになりました（図0-1）。「勉強」と聞くと身構えてしまうかもしれませんが、他の教科と違ってプログラミングにはパソコンを使います。パソコンを使った勉強なら、なじみやすいのではないのでしょうか。「勉強しろ」と言われるとやらされている感じがしていやいやの勉強になってしまうかもしれませんが、プログラミングを使うとこんなにいいことがあります。



図 0-1 勉強 ???

## ・色々な作業を自動化できます

プログラミングによって、これまで手動で行っていた作業を自動で行えるようになります。例えば1時間ごとに気温を測って記録する宿題があっても、プログラミングを使えば時間が来ると、自動で測って記録してくれます。

## ・生活が便利になります

昔のテレビはテレビを映すだけしかできませんでした。しかしプログラミングをはじめとした技術の発達により、テレビをネットにつなげることで、テレビでインターネットを使うなど様々な機能が増えました。他にも自動お掃除ロボット（図0-2）では、プログラミングによって掃除するコースを覚えていて、家の隅々まできれいにしてくれます。

## ・仕事になる

世の中では、プログラミングができる人を欲しがっている会社がたくさんあります。プログラミングができる人(プログラマーと呼びます)になることで、将来なりたい仕事の選択肢を増やすことができます。

## ・ゲームを作れる

RPGも格闘ゲームも、すべてのゲームはプログラミングによって作られています。欲しいゲームを買ってプレイすることは一般的ですが、プログラミングをマスターすると、自分がやりたいゲームを自分で作ってプレイすることができます。

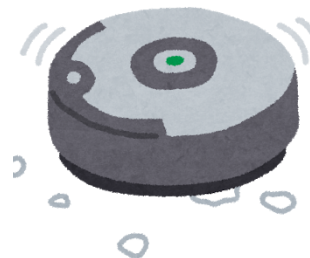


図 0-2 掃除ロボット



そんなプログラミングですが、実はプログラミングの種類はこの世に数千種類あると言われていて、それぞれプログラミングのルールが独自に決められていて、日本語や英語のように、同じ内容を異なるルールでプログラミングをします（図0-3）。

このことからプログラミングルールを総称して「プログラミング言語」と呼びます。古くからあるマシン語やC言語から、流行のPythonやLabVIEWまで様々です。どれがどれより優れている、と一概にいうことはできませんが、世の中には流行があるので、ある言語は廃れて、ある言語は人気があるなんてことがよくあります。どの言語を勉強するかはとても重要で、人気のない言語より人気のあるほうを勉強したほうが、世の中で必要とされるため勉強の価値が上がります。プログラミング学習のスタートは、どのプログラミング言語を選択するかから始まります。

本書では、プログラミング言語の一種であるLabVIEWを用いて、プログラミングを学びます。LabVIEWは単なる学習用のプログラミング言語ではなく、社会で一般的に使われています。会社ではLabVIEWを使用して新製品を開発したり、大学では最先端の研究を行ったりしています（図0-4、図0-5）。本書ではプログラミングに関する非常に細かいところ（例えばビット演算など）は解説していませんが、LabVIEWによる「測る」「制御する」技術を学んでいただき、純粋にプログラミングとサイエンスを楽しんで頂きたいと考えています。

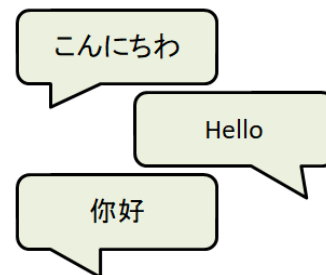


図 0-3 あいさつ

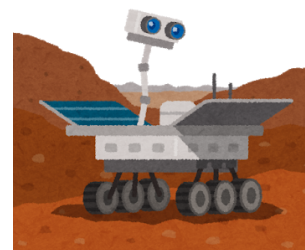


図 0-4 火星探査ロボット



図 0-5 IoT（モノのインターネット）

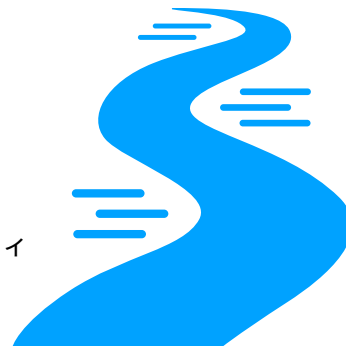
# 目次

## 序文

LabVIEWの生みの親であるJeff Kodosky氏からのメッセージ

## はじめに

本書は、プログラミングで自分の趣味を広げたいという方、Arduinoを使ったフィジカルコンピューティングやプロトタイピングに興味にある方などに向けてLabVIEWを一から学ぶことが出来るような内容を目指しました。中学生以上の読解力があれば読みこなせるような文章を心がけました。



## 第1部 LabVIEWを始めよう

プログラミングは、身近な問題を整理して、試してみて、さらに考えを進めるために役に立つでしょう

### 第1章 LabVIEWで学ぶプログラミング

コンピュータ活用の歴史を振り返ってLabVIEWの位置づけについて説明します

- 1.1 コンピュータとプログラム . . . 1
- 1.2 テキスト型言語とグラフィカル言語 . . . 3
- 1.3 フィジカルコンピューティングとLabVIEW . . . 4
- 1.4 LabVIEWの活躍している世界 . . . 6
- <コラム1 LabVIEWの向かう先はNXG>

### 第2章 LabVIEWコミュニティ版を使ってみよう

LabVIEWの特徴、本書の構成、LabVIEW Community Editionのインストール手順を示します

- 2.1 LabVIEWとLabVIEW NXGの特徴 . . . 9
- 2.2 本書の構成について . . . 11
- 2.3 LabVIEW Community Editionのインストール . . . 12
- <コラム2 LabVIEW NXG Community Editionのインストールについて>

## 第2部 LabVIEWプログラミング入門

LabVIEWプログラミングの基本を説明します

### 第3章 グラフィカルプログラミングにふれる

アイコン、ワイヤー、データフローなどLabVIEW特有のプログラミング方法を説明します

3.1 LEDの特性をLabVIEWで考えよう	・・・22
3.2 LabVIEWプログラムをのぞいてみる	・・・24
3.3 簡単なLabVIEWプログラムを作る	・・・26
3.4 繰り返し実行されるようにする	・・・35
<コラム3 LabVIEW NXGでのプログラミング>	

### 第4章 自分のアプリを作ってみる

データ収録にもつながる録音、再生プログラムで実用性のあるアプリケーションの作り方を説明します

4.1 趣味に活かすLabVIEWプログラミング	・・・43
4.2 “SoundVIEW”を操作してみる	・・・45
4.3 録音と再生のサンプルプログラム	・・・48
4.4 波形データと配列	・・・51
4.5 Forループとシフトレジスタと配列	・・・54
4.6 波形生成とサブVI	・・・57
4.6 録音、再生プログラム	・・・61
<コラム4 LabVIEW NXGのWEB親和性>	



## 第3部 LabVIEWとArduinoで電子工作

Arduino UNO、ブレッドボード、タクトスイッチ、ファン、LED、固定抵抗、可変抵抗、ワイヤー6本を使います//電子工作が初めての人は「みんなのArduino入門」の基本キットがお勧めです

## 第5章 LabVIEWとArduino

Arduinoを使ってフィジカルコンピューティングを始めます

5.1 Arduino IDEのインストールとLチカ	・・・70
5.2 Arduinoの入力と出力	・・・72
5.3 LabVIEWでArduinoをコントロール	・・・73
5.4 スイッチカウンターを作る	・・・76
5.5 ファンコントローラを作る	・・・78
5.6 スイッチの動作を変えよう	・・・83
<コラム5 LabVIEW NXGとハードウェア>	

## 第6章 LEDの特性を調べてみよう

LEDの特性を調べます

6.1 LEDの電圧電流特性の実験回路の組み立て	・・・87
6.2 LEDの電圧と電流の測定	・・・88
6.3 LEDの電圧電流特性曲線の表示プログラム	・・・91
6.4 LEDのV-I特性データの回帰分析	・・・92
6.Appendix 可変抵抗器や実験回路についての補足説明	・・・96
<コラム6 LabVIEW NXGでデータ解析>	

## 第7章 半導体センサを使ってみる

スマートフォンや自動車などに使用されているセンサがArduinoに接続しやすいモジュールとして発売されているので使い方の一例を紹介します

7.1 スマートフォンや自動車に使われている半導体センサ	・・・101
7.2 心拍数を測定するセンサ	・・・102
7.3 Arduino用サンプルプログラムで動作確認	・・・102
7.4 データシートを手に入れよう	・・・104
7.5 LabVIEWシリアル受信プログラムの作成	・・・106
7.6 心拍数測定プログラムの作成	・・・112

<コラム7 LabVIEW NXGの目指すもの>

## サンプルVIのリスト

あとがき

制作記録

# 第 1 部

## LabVIEWを始めよう

### 第 1 章

## LabVIEWで学ぶ プログラミング



コンピュータとプログラムについて簡単に振り返ってグラフィカル言語LabVIEWの特徴とLabVIEWが使われている世界を紹介します。

**[キーワード]** プログラミング、テキスト言語、グラフィカル言語、LabVIEW、フィジカルコンピューティング、Arduino、Raspberry Pi、LINX、LabVIEW Community Edition

### 1.1 コンピュータとプログラム

そもそもプログラムとは何でしょう。プログラムは、パソコンに仕事をお願いする際の手順書です。例えば「今から言うことをメモしてね」

「まずトマト100円、お肉200円、それと本がえっと…2冊だから2000円と書いておいてね」とお願いされたとしましょう。

この時のメモを見ると、こんな感じです (図 1-1)。

しっかりお手伝いできましたね。



図 1-1 買い物のメモ



では同じようにパソコンにもお手伝いをお願いしたいと思います。

その結果なんと、なにもメモがされていません (図 1-2)。

一体どうしてでしょう。

実は人に頼むのと同じようなお願いの仕方では、パソコンはお手伝いできないのです。私たちはこのお願いを以下のような手順に分解して理解できません。

- (1) メモを取るの、鉛筆とメモ帳を準備しよう
- (2) 「まずトマト100円」って言ったから、「トマト 100円」って書こう
- (3) 「お肉200円」って言ったから、「お肉 200円」って書こう
- (4) 「それと本がえっと…」は「本」ってメモしよう
- (5) 「2冊だから2000円云々」は、「本 2000円」って書こう

という具合にです。一方パソコンは、そもそも「メモをする」というお願いがわかりません。仮にメモがわかっても、まず鉛筆とメモ帳を準備する必要があることを知りません。パソコンにお手伝いしてもらうためには、手順を事細かに教えてあげる必要があります。パソコンが理解できるお願いは私たちに比べると非常に少ないため、お願いをする際にはパソコンがわかるようにかみ砕いて説明してあげる必要があります。



図 1-2 パソコンのお手伝い結果

このようにパソコンにお願いしたい手順を作成することをプログラミングといいます。

さらに大変なことに、パソコンには決められた順序で物事をお願いしなくてははいけません。

「鉛筆を使って文字を書いてね」と「文字を書くには鉛筆を使ってね」は、私たちにとっては同じ内容でも、パソコンからするとこの微妙な違いを理解できません。パソコンへのお願いの仕方はきっちり決まっていて、これを少しでも間違えるとパソコンは作業ができなくなります。

さらにさらに、せっかく手順を書いてもそのままではパソコンに伝わりません。私たちの言葉とパソコンが分かる言葉は実は違います。「文字を書いてね」の一言も、パソコンから見ると「なに言ってるの？」って感じです。そこでパソコンが読めるように、手順をパソコン語に翻訳する必要があります。これをコンパイルと言って、プログラミングによって作成された手順書をコンパイルするとようやく、パソコンは仕事を始められます (図 1-3)。

## 1.2 テキスト型言語とグラフィカル言語

プログラミングの世界では、多くの場合「こんにちは」とパソコンの画面に表示することを最初の導入とします。数千種類のプログラミング言語は、「テキスト言語」と「グラフィカル言語」の2つに大きく分けることができます。

### テキスト言語 - C言語の例

```
#include <stdio.h> //画面の表示機能を使うための準備
int main(){ //プログラムの始まり
    printf("Hello, World"); //こんにちはと挨拶
    return 0; //プログラムの終了
}
```

テキスト言語では、メモ帳などにキーボードで文字をタイプすることでプログラミングを行います。この例ではprintfという記述が、パソコン画面に「Hello, World」という文字を表示するための命令で

す。テキスト言語では、お願いに対してそれぞれ記述ルールが決まっています。プログラマはそのルールに沿ってお願いを順番にタイプしていきます。実行のためにはコンパイルが必要で、コンパイルをしたのち実行すると、プログラムはメモ帳に書かれた順番、上から下へ順番に命令を実行していきます。



図 1-3 プログラミング

### グラフィカル言語 - LabVIEWの例

グラフィカル言語では、画面に様々なアイコンを配置し、アイコン同士を線でつなげることによってプログラミングを行います（図 1-4）。

この例では、「Hello, World」と書かれたアイコンと、「文字列」アイコンを線でつなげることで、プログラミングを行います。実行すると右側の画面に「Hello, World」と表示がされます。

ほとんどのグラフィカル言語は専用のプログラミングソフトが提供され

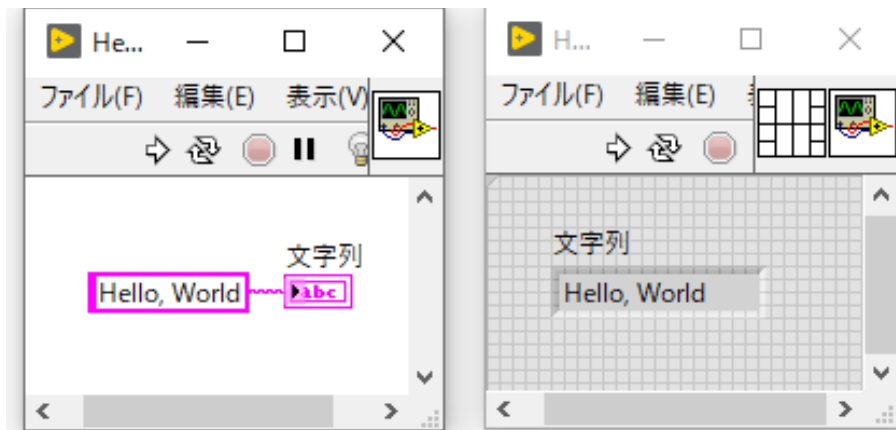


図 1-4 グラフィカル言語 - LabVIEW

ていて、それを用いてプログラミングを行います。プログラミングソフトのことを開発環境と呼びます。グラフィカル言語ではパソコンへのお願いに対してそれぞれアイコンが用意されていて、プログラマはアイコンと配線によってプログラミングを進めていきます。開発環境に専用の実行ボタンが用意されていて、ボタンを押すとコンパイルから実行まで自動で行われます。

テキスト言語とグラフィカル言語では、どちらが優れているといったことはなく、どちらも一長一短があります。例えばホームページを作る作業はJavaScriptなどのテキスト言語が得意とし、LabVIEWは苦手です。逆に気温を5分毎に測るような作業をプログラミングで行う際は、LabVIEWが最も得意とする分野です。世の中で使われているプログラミング言語のほとんどはテキスト言語に属します。しかしLabVIEWやSimulink(Mathworks社の製品です)などのグラフィカル言語は、自動車業界など特定の分野で非常に活発に使用されています。

## 1.3 フィジカルコンピューティングとLabVIEW

プログラミングを生活の中で意識するケースは多くないと思います。「あー、今日プログラミング感じたなー」という日はあまりありませんよね。プログラミングがそれほど意識しにくいのは、普段の生活で意識して使う機会がほとんどないからです。そこで海外の人が、「もっとパソコンやプログラミングを身近にして、パソコンの可能性を広げましょう」という考え方、**フィジカルコンピューティング**を考えました。このキーポイントは、「パソコンやプログラミングを身近にする」ということです。これは毎日プログラミングを勉強しましょうという意味ではなくて、生活の中で良くしたいところを、パソコンとプログラミングで解決していこうよ、という意味です。例えば電気の消し忘れが気になったら、パソコンといろいろな電子工作を使って、自動で電気をOFFにするというカラクリを作ることができます。最近のゲーム機では専用のリモコンを振って遊べますが、これもフィジカルコンピューティングの一種で、リモコンの中に入っている小さなパソコンで、リモコン内の加速度センサなどの情報を読みとり、ゲーム画面という形で人に情報を見せています。このように普段の生活の中にパソコンとプログラミングを取り入れることは、プログラミングを勉強する観点からもとても大切です。勉強のために勉強するプログラミングは絶対に身につかず、何か目的をもって勉強することで、身につくスピードは格段に早くなります。LabVIEWは、電子工作で作ったものを取り扱うことが非常に得意です。LabVIEWと電子工作を組み合わせることでパソコ

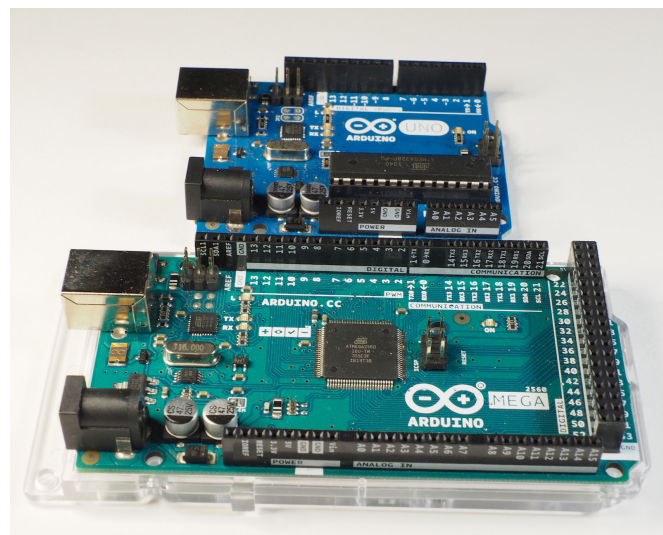


図 1-5 Arduino Mega 2560（手前）と Arduino UNO（奥）



ンだけの世界から飛び出し、気温を測ったり電気スイッチをON/OFFしたりできます。LabVIEWはフィジカルコンピューティングの考え方に最もマッチしたプログラミング言語かもしれません。

パソコンと私たちの世界をつなぐためには、電子工作で作られた製品、すなわちハードウェアが必要です。これまではハードウェアといったらとても高価で、プライベートで買うことは難しかったのですが、今は「シンプルなものをもっと安価に」という考えが広がっていて、手ごろな価格でハードウェアを購入し、プログラミングなどで遊ぶことができます。LabVIEWから使いやすいハードウェアを2種類ご紹介します。

ひとつは**Arduino** (図 1-5) で「アルドゥイーノ」と呼びます。

Arduino UNO は大体3000円くらいで購入できるハードウェアで、その手軽さと使いやすさから圧倒的な人気を集めています。電圧の入出力機能を持っているので、センサが出力する電圧を監視したり、LEDを光らせたりすることができます。ちなみにセンサというのは状態によって出力する電圧が変わる電子部品で、例えば光センサーではセンサにあてる光の強さによって電圧の大きさが変わります。この電圧の大きさを監視することで、今の光の強さを知ることができます (図 1-6)。

もうひとつは**Raspberry Pi** (図 1-7) で「ラズベリーパイ」と呼びます。これは7000円程度で購入できるハードウェアですが、Arduinoと違いOSを載せることができます。OSとは、「Windows」や「Mac」のように、パソコンを動かすためのソフトです。つまりRaspberry Piはパソコンです。なんと1万円以下でパソコンが買えるのです。もちろんすごく高性能なパソコンではないので、最新ゲームで遊ぶことはできませんが、OSが載せられることによってArduinoより複雑な作業をこなすことができます。

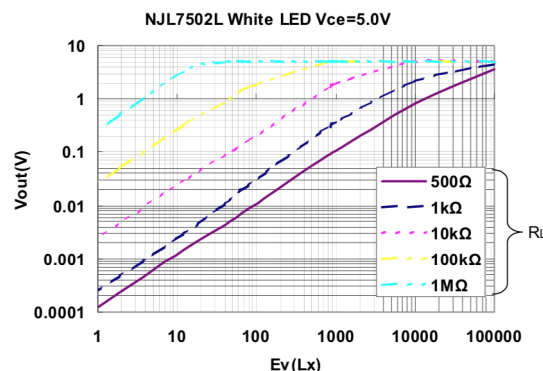


図 1-6 光センサ

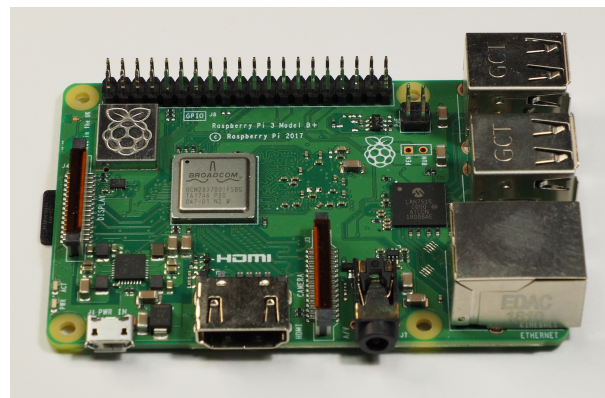


図 1-7 Raspberry Pi

なぜこの2つがLabVIEWから使いやすいかというと、この2つのハードウェアを使うための専用ツールをLabVIEWが持っているからです。実はハードウェアをパソコンとつなげてコントロールするためには、とても複雑で大変な作業をしなくてはなりません。でもLabVIEWは、**LINX(リンクス)**と呼ばれる専用ツールによってそれらの作業をスキップできるようになっています。このおかげで、私たちは本当に作りたいものを作ることに集中することができます。

そういえばLabVIEWの値段はいくらでしょう？ホームページでLabVIEWの値段を調べると、数万から数十万だと記載があります。ですがご安心ください、タダです。これまでは会社で使うときも遊びで使うときも、必ずLabVIEWを購入する必要があります。2020年からNational Instruments社の方針が変わり、非商用の場合、つまりお金儲けでなく個人的な趣味で使う場合だけは、誰でも無料でLabVIEWを使えるようになりました。これは**LabVIEW Community Edition**と名前がついていて、Community版と有料版の機能的な違いは全くありません。

繰り返しになりますが、LabVIEWでお仕事をする場合には、この無料版LabVIEWは使えず、きちんとLabVIEWの使用権(ライセンスといいます)を購入する必要があります。加えてLabVIEWが持つ様々な追加ツール(モジュールやアドオンと呼ばれる)を使用する際には、現状はライセンスの購入が必要です。

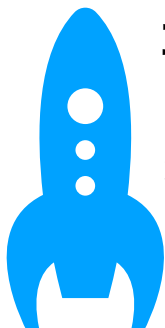
## 1.4 LabVIEWの活躍している世界

改めてLabVIEWの得意分野を紹介します。少し難しい話になりますが、プログラミング学習には直接関係しないので、ここは読み飛ばしてもらっても構いません。

LabVIEWはハードウェアと組み合わせての**自動検査**や**制御**を得意とします。ここでのハードウェアとは、オシロスコープやファンクションジェネレーターといった「計測器」から、カメラやモーターといった身近なものまで様々です。例えばある装置の生産工場では、装置が間違いなく出来上がったかの確認を、LabVIEWを用いています。製品に電圧をかけ、製品に流れる電流を測ることで、製品に問題がないかを確認しています。別の例では、袋に印刷された文字が間違っていないか、カメラを使った検査システムを運用しているケースもあります。このようにLabVIEWは様々なハードウェアと組み合わせることで、様々な会社で便利に使われています。

実際の事例については、以下のサイトから見れます。

<http://www.ni.com/innovations-library/case-studies/ja/> (左側の「業界」から興味のある業界を選択してください。)



## コラム 1 LabVIEW NXGの向かう先はNXG

National Instruments社は、2017年にLabVIEWの次世代バージョン「LabVIEW NXG」を発表しました。LabVIEW NXGはこれまでのLabVIEWをベースとせずに一から開発されたソフトウェアです。LabVIEWの開発手順はそのままに、ユーザーが待望した画面の拡大縮小や、プロジェクト管理の改善、ウェブ開発の新機能など様々な要素が追加、刷新されました。2020年現時点ではLabVIEWが持つ一部の機能しか備わっておらず、LabVIEW NXGはまだ発展途上です。National Instruments社はこれからも引き続きLabVIEW NXGの開発に投資すると明言しており、将来的にはLabVIEW NXGが主流となることが推測されます。

「え？ NXGに移行するからNational InstrumentsはLabVIEWを無料にしたのかつて？」  
そうではなさそうです。なんとLabVIEW NXGもCommunity Editionを無料で使えます。詳しくは第2章で説明します。

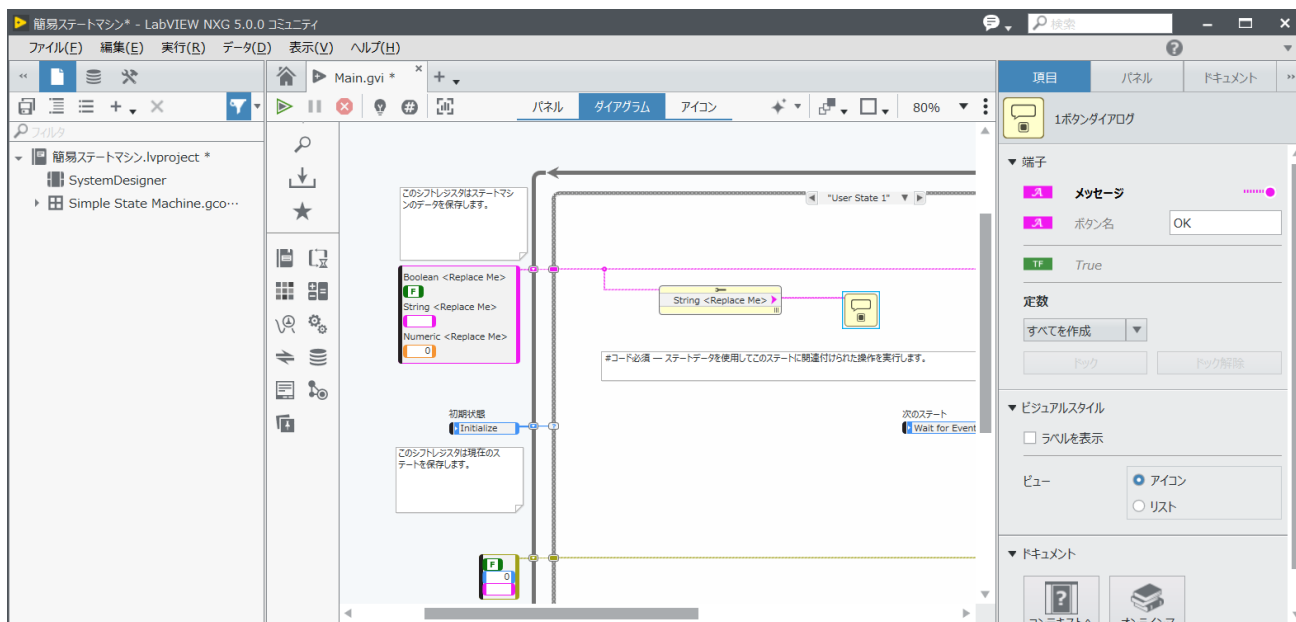


図 C1-1 LabVIEW NXGのプログラミング画面



# 第 2 章

## LabVIEW

## コミュニティ版を 使ってみよう



LabVIEWコミュニティ版（LabVIEW Community Edition）の紹介と本書の構成を説明します。さらにLabVIEW Community Editionのダウンロードとインストール方法について説明します。

**[キーワード]** LabVIEW Community Edition、LabVIEW NXG Community Edition、日本語版/英語版、Arduino、ユーザープロファイルの作成、LabVIEW Community Editionのダウンロード

### 2.1 LabVIEWとLabVIEW NXGの特徴

初めてLabVIEWが世の中に現れたのは、1986年です。当時はテキスト言語で開発することが普通だったので、エンジニアはまずテキスト言語を学び、ある程度学び終わってから、実際の仕事を進めていました。しかしこれでは勉強に時間がかかり、本来の仕事を素早く進められません。そんな中、アメリカでLabVIEWの最初のバージョンが発明されました。テキスト言語が主流だった当時、LabVIEWはあまり流行らなかったのですが、グラフィカルプログラミングの考え方が徐々に受け入れられていくと、LabVIEWはグラフィカルプログラミングの代表格となりました。グラフィカルなプログラミングはテキスト言語に慣れていないエンジニアでもすぐに学ぶことができ、それによって本来の仕事に集中できるようになったからです。それから約35年の間、LabVIEWは大きな機能追加を何度も重ねてきました。ユーザーからの声を大切に、ユーザーが提案して作られた機能もたくさんあります。LabVIEWは多くのエンジニアにとって必須ツールとなりました。

LabVIEWの特徴は「アイコン」と「ワイヤー」です。LabVIEWでは以下のように「関数ノード」と呼ばれるアイコンを画面上に置くことでプログラムを作成します（図 2-1）。



関数ノードは一つ一つが役割を持っていて、1ボタンダイアログ関数ノードは、「画面にメッセージを表示する」という機能です。関数ノードをお手伝いに例えると、「買う」や「書く」などの、作業をお願いすることと同じです。さらにLabVIEWでは、データの受け渡しをワイヤーと呼ばれる線を用いて行います。ここで「データ」とは、パソコンに作業をお願いする際に必要な情報のことです。お手伝いの例えでは、「何を」という作業の対象を示します。この例ではパソコンに対して、「こんにちは」(データ)を表示してね(作業)というお願いをしています(図 2-2)。

LabVIEWではこのように、関数ノードを置いて、ワイヤーを使ってお互いをつなげることでプログラミングをします。詳細については第3章でまた説明します。

LabVIEW Community Editionには、LabVIEWとLabVIEW NXG両方のバージョンがあります。現時点ではLabVIEWのバージョンは英語版しかありません。LabVIEW NXGは元々英語と日本語を切り替えることができますので、日本語でプログラミングを行うことができます。本書では、LabVIEWを使ってプログラミングを行います。コラムでもお話ししたようにLabVIEW NXGはまだ発展途上であり、LabVIEW NXGの開発が進んでいけばこちらがメジャーになると思いますので、その時には本書もLabVIEW NXGを使うように更新しようと思っています。

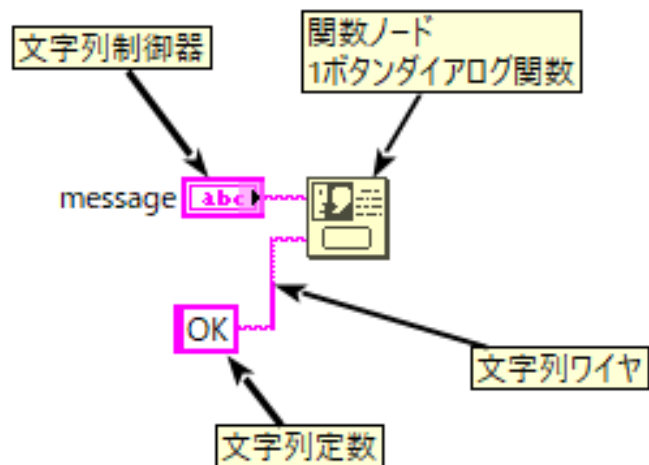


図 2-1 関数ノード

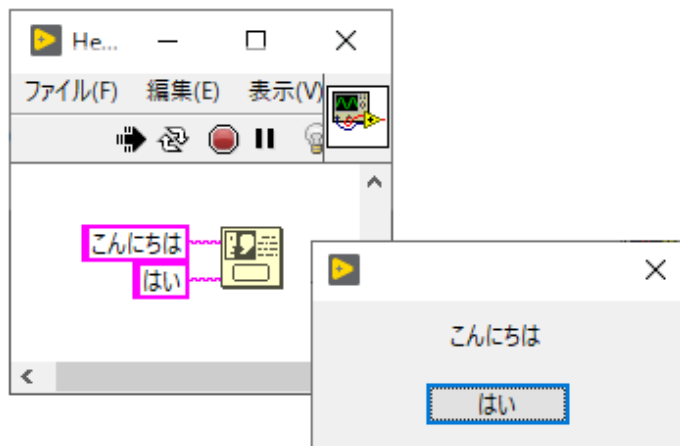


図 2-2 「こんにちは」と表示

## 2.2 本書の構成について

本書は以下のような方に向けて作られています。

- (1) 中学生以上
- (2) プログラミングをやってみたい、またはプログラミングをもっと身近に感じたい
- (3) 電子工作も少しやってみたい

また、全体は**3部構成**でそれぞれいくつかの章に分かれています。

- (1) **第1部**はLabVIEWの紹介とLabVIEWを使うための準備を行います。
- (2) **第2部**はPCだけがあればLabVIEWのプログラミングが学べる題材を用意しました。第3章では「LEDと電流制限抵抗」という簡単なアプリケーションを動かしながらLabVIEWの操作方法とプログラミングの方法を学びます。実際に四則演算や数値の大小比較などを学びながらLEDの特性に応じた電流制限抵抗の計算プログラムを作成し、プログラムの流れを考えたLEDシミュレータのようなアプリケーションを作ります。
- (3) 第4章ではPCのマイクやPCのアプリケーションで発生させた音を保存してスペクトログラムという周波数分析を行う**SoundVIEW**というアプリケーションを操作し、プログラムの内容を概観します。音声データをグラフに表示したり、逆再生するプログラムを作りながら、データを取り込んで処理する方法を学びます。
- (4) **第3部**では、入手性が良い「みんなのArduino入門」という書籍用の基本キットを前提にプログラミングを学びます。Arduino UNOと代用できる部品を持っている方はそれを使ってください。
- (5) 第5章の最初にArduinoの使い方を学びます。プッシュスイッチとDCファンを使ってデジタル入力とPWM出力を体験します。
- (6) 第6章ではLEDと抵抗と可変抵抗を使ってLEDのV-I特性を測定するプログラムを作成します。データ解析の例として直線回帰を使ってLEDのPN接合の係数を求めます。
- (7) 第7章ではArduinoをMAX30102モジュールのインターフェースとして血流データを取り込んで心拍数を求めるプログラムを作成します。I2C通信を扱います。またピンヘッダのはんだ付けが4カ所必要になります。

## 2.3 LabVIEW Community Editionのインストール

LabVIEW Community Editionのインストール手順をご紹介します。まずはNational Instruments社のユーザプロフィールを作りましょう。途中に個人情報を入力するところがあるので、未成年の人は保護者と一緒に作業をしてください。

1. National Instrumentsのホームページへアクセスします。  
<https://www.ni.com/ja-jp.html>
2. 右上の「ログイン」をクリックします（図 2-3）。
3. 下のほうにある「ユーザプロフィールの作成」をクリックします（図 2-4）。
4. 必要な情報を入力して、「ユーザプロフィールの作成」をクリックします（図 2-5）。



図 2-3 ログイン画面

### ユーザプロフィール ログイン

A login form for a user profile. It features a placeholder for a profile picture, followed by input fields for '電子メール' (Email) and 'パスワード' (Password). Below the password field is a link: 'パスワードをお忘れになった場合' (If you forgot your password). At the bottom, there is a checkbox labeled 'ログイン状態を維持' (Keep me logged in) and a blue 'ログイン' (Login) button. A red box highlights a link at the bottom: 'ユーザプロフィールの作成 >' (Create user profile >).

図 2-4 ユーザプロ  
ファイルの作成

### ユーザプロフィールの作成

既にアカウントをお持ちですか？ [ログイン](#)

A form for creating a new user profile. It includes input fields for '氏(漢字)' (Surname in Kanji), '名(漢字)' (First name in Kanji), '氏(かな)' (Surname in Kana), and '名(かな)' (First name in Kana). The '氏(漢字)' field is marked as '必須項目' (Required). There is a dropdown menu for '職種' (Occupation) with '指定してください' (Please specify) selected. Below these are input fields for '電子メールアドレス' (Email address) and 'パスワード (半角英数)' (Password, alphanumeric). At the bottom, there is a blue button labeled 'ユーザプロフィールの作成' (Create user profile).

図 2-5 ユーザプロファ  
イルの作成（入力）

ユーザプロファイルの作成が完了したら、いよいよLabVIEW Community Editionをインストールしましょう。

1. このドキュメントのダウンロードサイトから、LabVIEW Community Editionへのリンクをクリックします (図 2-6)。

<http://www.quatsys.com/labview/1109/lvproraku.jp.html>

2. 「DOWNLOAD NOW」 から、「LabVIEW 2020 Community Edition」 をクリックします (図 2-7)。

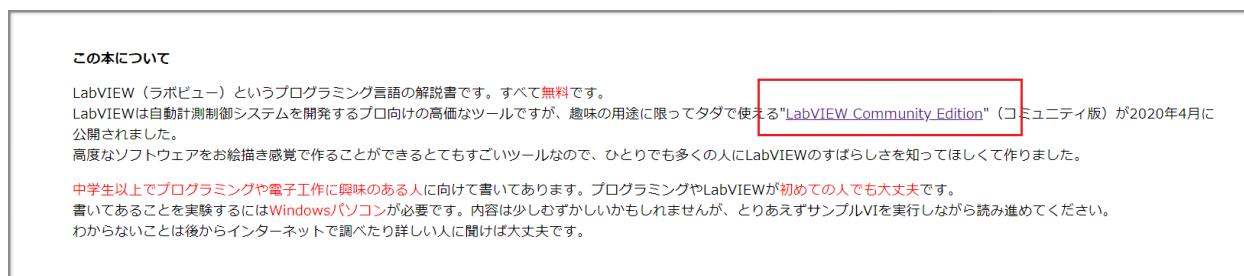


図 2-6 LabVIEW Community Editionへのリンク

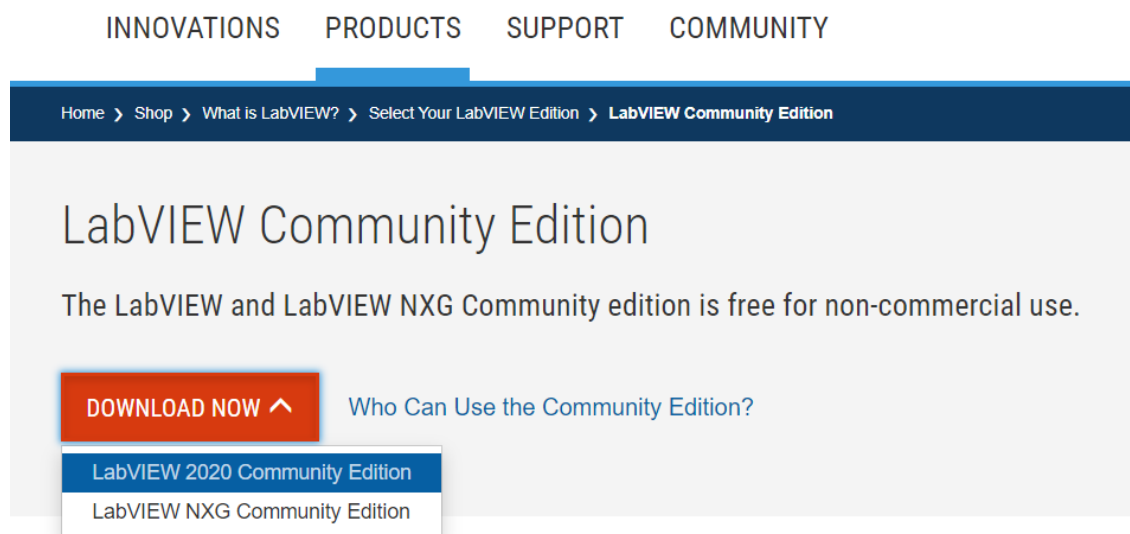


図 2-7 ダウンロード画面

3. ダウンロードサイトへ移動します。右側の「ダウンロード」をクリックします。ダウンロードには、作成したユーザプロフィールでのログインが必要です (図 2-8)。
4. ダウンロードするファイルの取り扱いについて聞かれますので、「保存」を選択して好きな場所に保存してください (図 2-9)。(Microsoft Edge以外のブラウザでは多少操作が異なりますが、ファイルはPCに保存してください。)

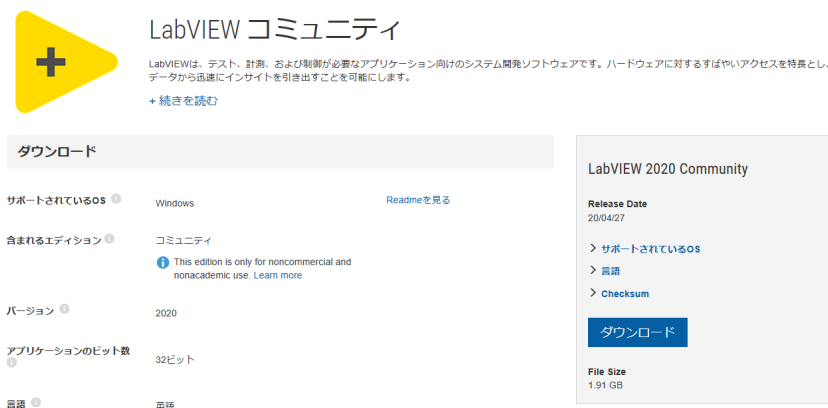


図 2-8 ダウンロードサイト



図 2-9 ダウンロード開始

5. ダウンロードが完了すると、選択された場所に「ni-labview-2020-community-x86\_xxx.iso」がダウンロードされます。右クリックして、「マウント」を選択します（図 2-10）。
6. マウントが完了すると仮想DVDドライブができて、ISOファイルに入っているファイルの一覧が表示されます。「Install.exe」をダブルクリックして実行します（図 2-11）。

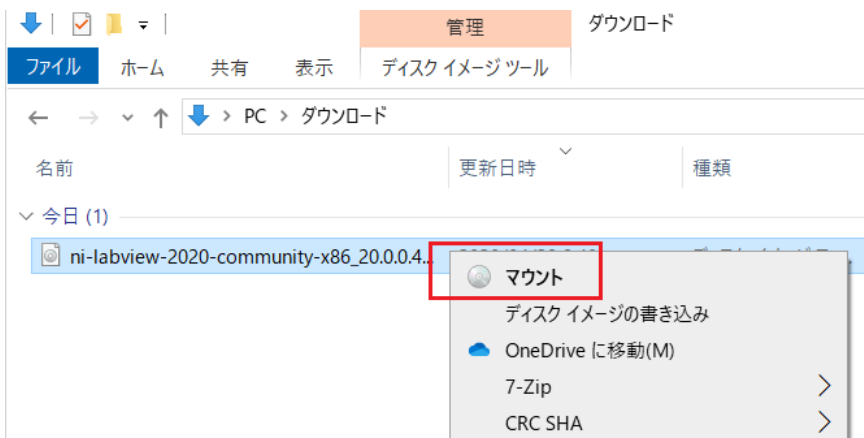


図 2-10 ISOファイルのマウント



図 2-11 インストールフォルダー一覧

- 「Install.exe」を実行すると、まずはソフトウェアの管理に必要な「NIパッケージマネージャ」がインストールされます。ライセンス契約書を読んで、同意する場合は「上記のライセンス契約書に同意します。」を選んで次へ進んでください（図 2-12）。どんどん次へ進むと、インストールが始まります（図 2-13）。
- NIパッケージマネージャのインストールが完了すると、そのままLabVIEWのインストール画面へ移動します。ここでLabVIEWの追加項目を選択します。今回はすべてを選択した状態で次へ進んでください（図 2-14）。

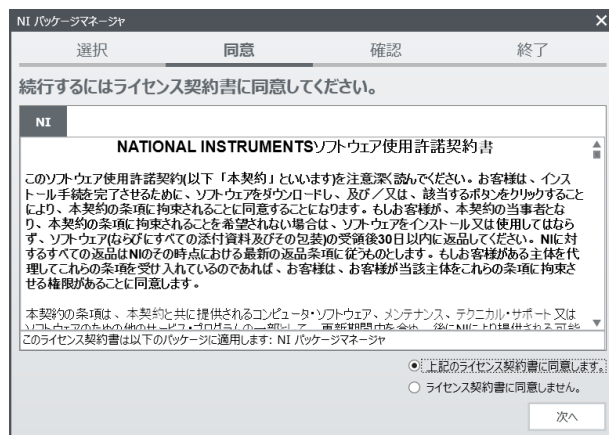


図 2-12 NIパッケージマネージャライセンス契約書

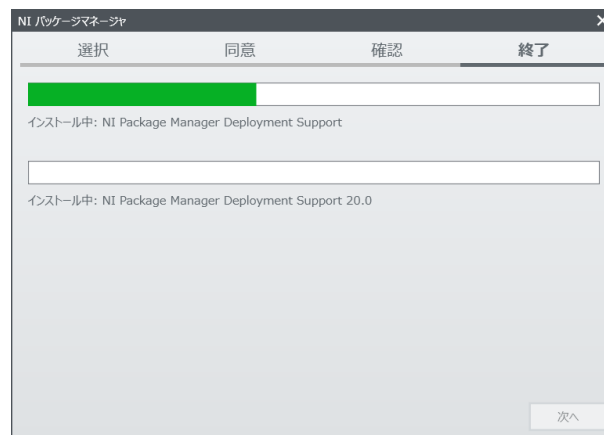


図 2-13 NIパッケージマネージャインストール開始



図 2-14 LabVIEWインストール項目選択画面

9. NIパッケージマネージャの時と同じように、ライセンス契約書が表示されます。きちんと読んで、同意する場合は次へ進んでください（図 2-15、図 2-16）。
10. 最後にもう一度、インストールする項目に間違いがないか聞かれます。次へ進むと、インストールが始まります（図 2-17）。
11. インストールが完了すると、「NIカスタマエクスペリエンス向上プログラム認定」に参加するか聞かれます。どちらでもかまいませんので希望するほうを選んでOKボタンをクリックしてください（図 2-18）。



図 2-15 LabVIEWライセンス契約



図 2-16 マイクロソフトライセンス契約



図 2-17 インストール項目最終確認



図 2-18 NIカスタマエクスペリエンス向上プログラム認定



12. 続いて「アクティブ化」という作業が必要になります（図 2-19）。LabVIEWを使うためには、作成したユーザプロフィールでログインして、アクティブ化を行う必要があります。「ログインしてアクティブ化する」をクリックすると、別画面でログイン画面が表示されます（図 2-20）。ここからログインをしてください。
13. ログインが完了したら、アクティブ化をします。「アカウントのライセンスを確認」が選択されている状態で、「アクティブ化」をクリックしてください（図 2-21）。皆さんのアカウントには、すでにLabVIEW Community Editionのためのライセンスが追加されていますので、アクティブ化のために特別な作業は必要ありません（図 2-22）。
14. 最後にパソコンを再起動する必要があります。作業中のソフトウェアは保存するなどして一度終了したうえで、パソコンを再起動してください（図 2-23）。
15. 以上でLabVIEWのインストールは完了です。Windowsのスタートメニューから、LabVIEWがインストールされていることを確認してください（図 2-24）。

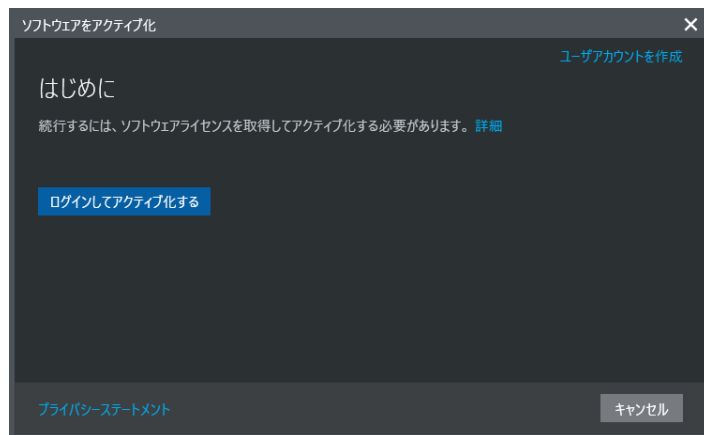


図 2-19 アクティブ化画面



図 2-20 ユーザプロフィールの入力

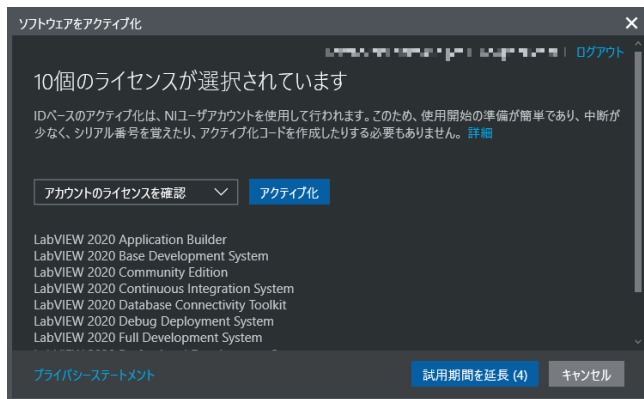


図 2-21 ライセンスのアクティブ化



図 2-22 アクティブ化完了



図 2-23 再起動

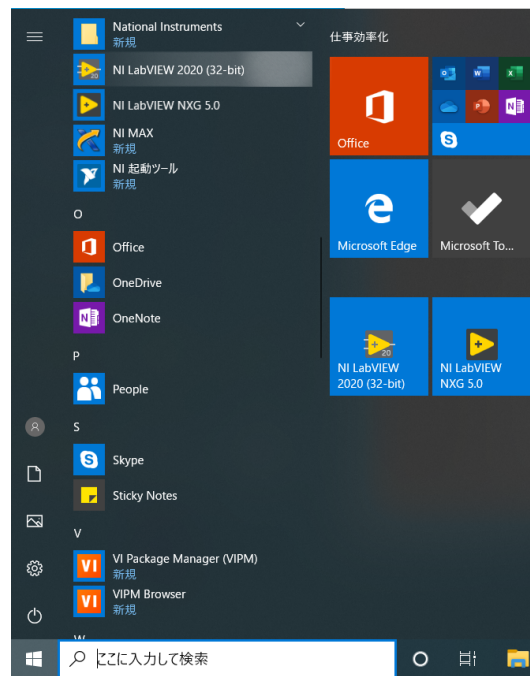
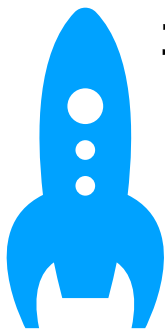


図 2-24 Windowsスタートメニュー



## コラム2 LabVIEW NXG Community Editionのインストールについて

LabVIEW NXG Community Editionのインストールも、LabVIEWのインストールとほとんど同じです。LabVIEW Community Editionへのリンクから、「LabVIEW NXG Community Edition」を選んで、インストーラを入手してください（図 C2-1）。



### LabVIEW NXG コミュニティ

LabVIEW NXGを使用すると、ハードウェアを迅速に自動化し、プロジェクトの仕様に合わせてテストをカスタマイズし、簡単に計測結果を表示できます。

[+ 続きを読む](#)

#### ダウンロード

サポートされているOS ①	Windows	<a href="#">Readmeを見る</a>
含まれるエディション ①	コミュニティ <b>i</b> This edition is only for noncommercial and nonacademic use. <a href="#">Learn more</a>	
バージョン ①	5.0	
アプリケーションのビット数 ①	64ビット	
言語 ①	ドイツ語, フランス語, 中国語, 日本語, 英語, 韓国語	

#### LabVIEW NXG 5.0 Community

Release Date  
20/04/27

> [サポートされているOS](#)

> [言語](#)

> [Checksum](#)

ダウンロード

File Size  
4.02 GB

図 C2-1 LabVIEW NXG Community Editionのダウンロードリンク

# 第 2 部

## LabVIEW

### プログラミング入門

---

# 第 3 章

## グラフィカルプログラ ミングにふれる



LabVIEWプログラミングの基本やLabVIEWの特有のプログラミング方法を説明します。

**[キーワード]** フロントパネル、実行ボタン、連続実行ボタン、停止ボタン、一時停止ボタン、制御器、表示器、ブロックダイアグラム、端子、（端子とワイヤーの）色、実行のハイライト、関数、ヘルプ、新規VI、制御器パレット、関数パレット、数値制御器・表示器、ブール制御器・表示器

### 3.1 LEDの特性をLabVIEWで考えよう

LED（図3-1）はLight Emitting Diodeの頭文字をとった名称で、電流を流すと光る部品です。電球と違って、長く使える、電気代が安く済む、スイッチを入るとすぐに光るといった良い点がたくさんあります。2014年には青く光るLEDの発明と実用化で、日本人の方3名が、ノーベル物理学賞というすごい賞を受賞しました。LEDはとても便利で、生活の中のあちこちで使われています。LEDの原理について紹介しながら、LabVIEWの使い方に慣れていきましょう。

LEDには向きがあり、プラスの方向からマイナスの方向へ電流を流す必要があります。足の長いほうをプラスの「アノード」、短いほうをマイナスの「カソード」と呼びます。アノード側の電圧を、カソード側より高くすると、電流がアノードからカソードの方向へ流れます。LEDの中にはプラスの電気とマイナスの電気が分かれて存在して、電流が流れるとそれが互いにつつき合い、ぶつかったエネルギーが光として私たちの目に見えます（図3-2）。

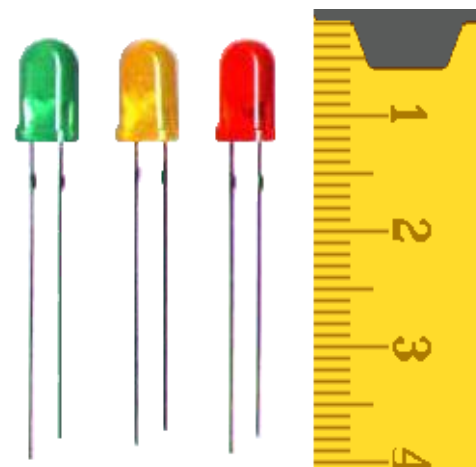


図 3-1 緑色LED、黄色LED、赤色LED

LEDを光らせるための回路を作ると、図3-3のようになります。LabVIEWを使えば、電気回路を作る代わりに、プログラミングで電気回路を再現できます。こうすると、ちょっと抵抗の値を変えるために抵抗部品を取り換えたり、電池を何個も準備したりする必要がありません。世間ではこのように、プログラミングで現実世界を再現することを「シミュレー

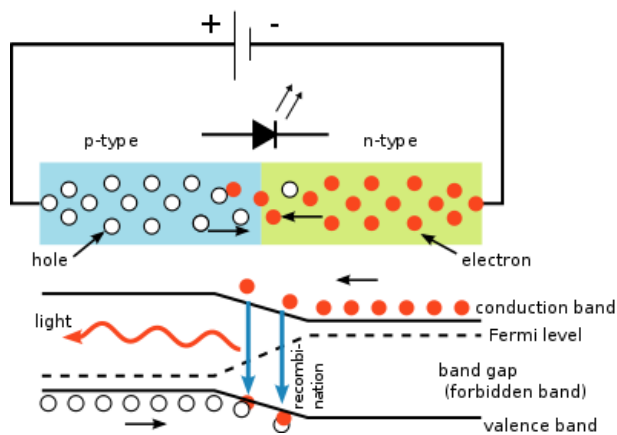


図 3-2 LEDの発光原理

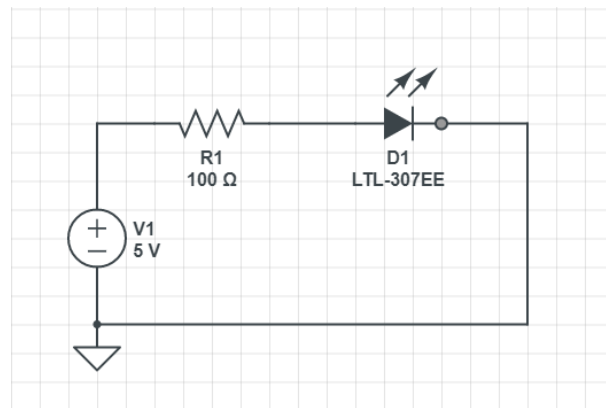


図 3-3 回路図

ション」なんて呼びます。自動車や飛行機の開発でもシミュレーションが使われています。

まずLabVIEWを起動します。Windowsのスタートメニューから、「NI LabVIEW 2020 (32bit)」を選択し、LabVIEWを起動します(図3-4)。そうしたら、3章のサンプルフォルダにある**LED Simulator.vi**を開きます。画面は図3-5のような見た目です。

さてまずはとりあえず、プログラムを実行してLEDの様子を確認してみましょう。LabVIEWではプログラムのことを**VI(ブイアイ)**と呼びます。画面左上にある実行ボタンをクリックしてVIを実行します。実行すると、LEDが赤色に点灯します(図3-6)。

最初の設定では、LEDはこのように点灯します。続いて「電流制限抵抗値」のダイヤルをマウスで回して、「500」付近にして実行してみてください。今度はLEDが少し暗くなったでしょうか。いろいろな値を試すのに毎回実行ボタンを押すのは大変ですね。実行ボタンの隣にある連続実行ボタンを押すと、VIを自動で繰り返し実行してくれます。止めるときにはもう一度連続実行ボタンを押します。

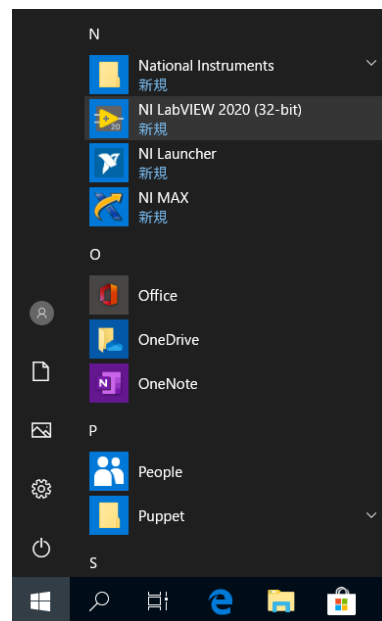


図 3-4 Windowsメニューからの起動

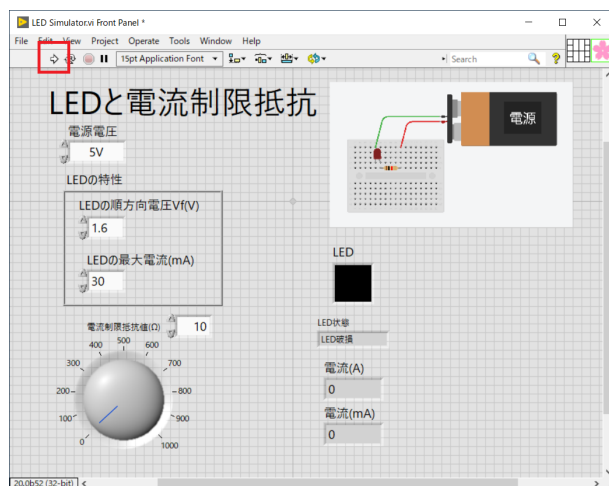


図 3-5 LEDと電流制限抵抗 フロントパネル

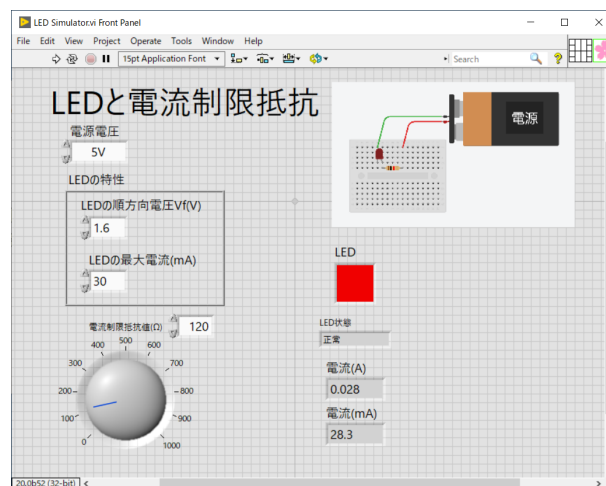


図 3-6 初期状態での実行結果

さてそれぞれの「モノ」について説明をします。「電源電圧」は電池の電圧を決めます。こういった電気工作では、5/3.3/1.8Vがよく使われるので、その3種類から選べるようにしています。「電流制限抵抗値」は、LEDに流れる電流の量を制限します。LEDに流れる電流は多ければ多いほど光が強くなるのですが、あまりに多すぎると、電気同士の衝突が強くなりすぎて、LEDが壊れてしまいます。それを防ぐために抵抗を使って、電流の量をコントロールします。この値が大きいほど、LEDに流れる電流は小さくなります。「順方向電圧」はLEDを光らせるために必要な電圧です。電圧があまりにも小さいと、電気同士はゆっくりぶつかるので、光るために十分なエネルギーが得られません。この電圧より大きな電圧をかけないと、LEDは光ることができません。そして「LEDの最大電流量」は、「これ以上電流が流れるとLEDが壊れますよ」という値です。例えば以下のような設定では、「LED破損」が表示され、LEDが壊れてしまったことがわかります（図3-7）。

もし間違った電気回路を作ってしまうと、LEDが壊れてしまうかもしれません。シミュレーションであれば壊れるような設定にしても、実際にLEDが壊れることはありません。新しい製品を開発するときなどは、このようなシミュレーション技術が頻繁に使われています。他にもいろいろな設定を試して、まずはLabVIEWの操作に慣れてください。

- (1) 「電流(mA)」の値がどのようになると、LEDは壊れますか？
- (2) 電源電圧を5Vから1.8Vにしたとき、5Vの時と同じ色を出すにはどうすればよいでしょう？  
(答えは1つではありません)
- (3) LEDが破損になった場合、どこを変更すれば破損を避けられるでしょう？

## 3.2 LabVIEWプログラムをのぞいてみる

では実際にLabVIEWプログラムがどのように作られているか

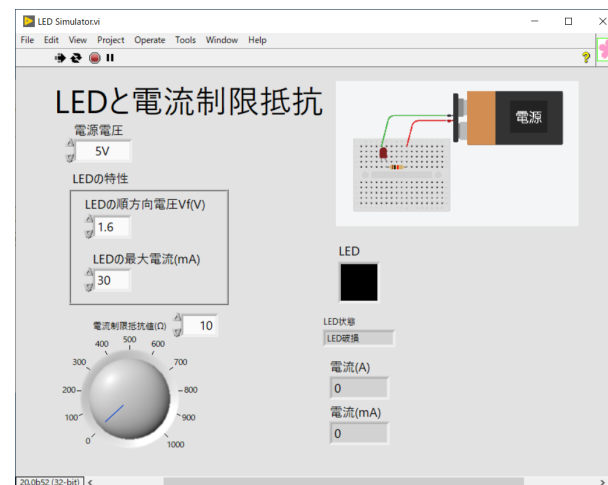


図 3-7 LEDの破損

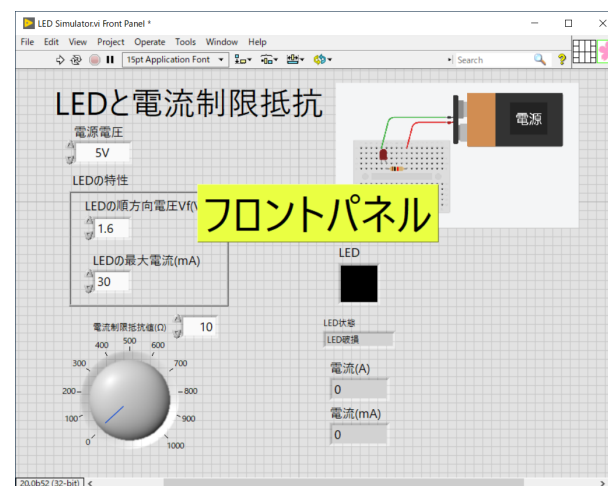


図 3-8 フロントパネル



見ていきましょう。皆さんがこれまで見てきた画面は**フロントパネル**と呼ばれ、設定を変更したり結果の値を確認したりする画面です。プログラミングの世界では、これを**ユーザーインタフェース(UI)**と呼びます。「インタフェース」とは、何かと何かを繋ぐときの懸け橋になるもので、この場合はユーザーとプログラムの懸け橋になるのがUIです(図3-8)。LabVIEW上部のWindow 》 Show Block Diagramを選択すると、背景が白い画面が表示されます。これを**ブロックダイアグラム**と呼びます。LabVIEWではブロックダイアグラムでプログラミングをし、フロントパネルからプログラムを使います(図3-9)。

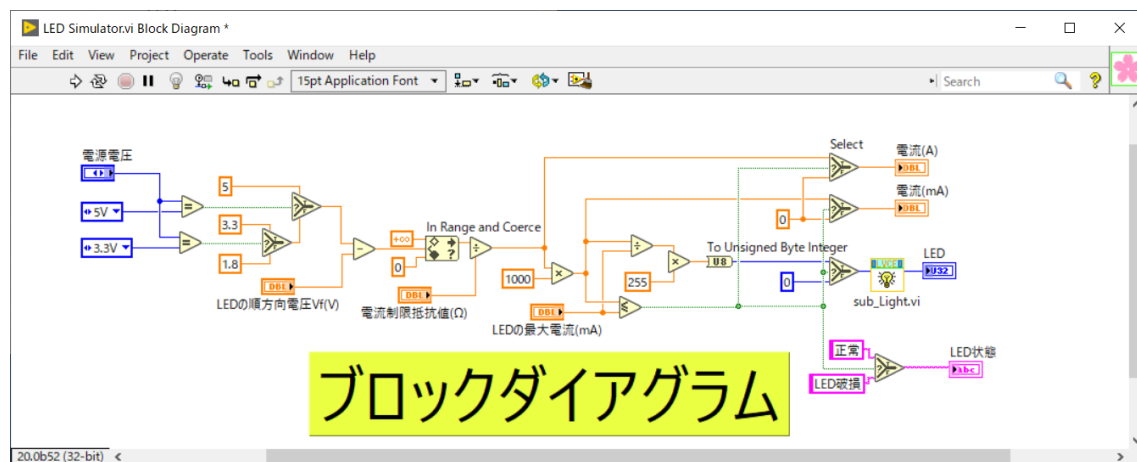


図 3-9 ブロックダイアグラム

ブロックダイアグラムを見ると、なんだかカラフルですね。LabVIEWでは、取り扱うデータによって色が決められています。数ならオレンジや青、文字はピンク、などのようです。よく見ると、フロントパネルにあるものと同じ名前のものがいくつかありますね。「端子」と呼びますが、フロントパネル上のほとんどのモノは、この端子とセットになっています。背景が薄い黄色のものは「関数」と呼んで、割り算や値の大小比較などの「何か」の作業をしてくれるものです。実際にどのようにプログラムが動くか見てみましょう。

**実行のハイライト** (図3-10) をクリックすると、電球マークが点灯します。このままVIを実行してください。プログラムがどのように実行されるか、アニメーションで表示されます。端子を見ると、フロントパネルで設定した数

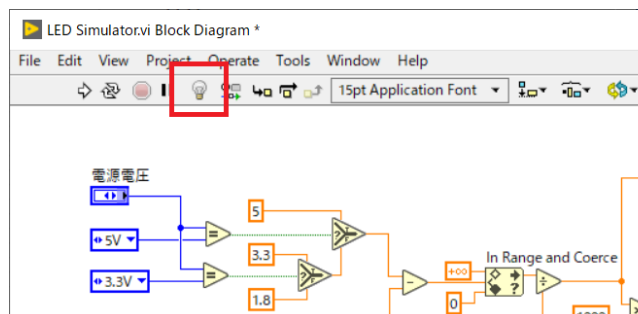


図 3-10 実行ハイライト

が表示されていて、フロントパネルのモノと端子はペアになっていることが確認できますね。

LabVIEWでは「端子」と「関数」を、「ワイヤ」で配線することによってプログラミングしていきます。この例では左から右へプログラムが実行されています。LabVIEWではワイヤをどのように配線するかで、実行の流れを右から左にも、上から下にもコントロールできます。ですが、左から右へ流れるようにプログラミングをすることが、**LabVIEWプログラムのマナー**として決められています。

動作確認が終わったら、フロントパネルの右上にある×ボタンを押してVIを終わります。ブロックダイアグラムの×ボタンは、ただブロックダイアグラムを閉じるだけなので、VIを終らせることはできません。図3-11のように保存するか聞かれたら、「Don't Save」を選んでください。

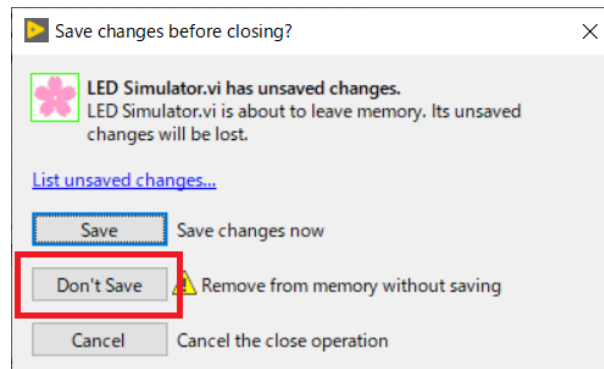


図 3-11 保存ダイアログ

### 3.3 簡単なLabVIEWプログラムを作る

では実際に、初めからVIを作ってみましょう。先ほどのLEDの例を、順を追って作っていきましょう。まずは開いているすべてのVIを閉じて、LabVIEWのスタートアップ画面を開きます（図3-12）。

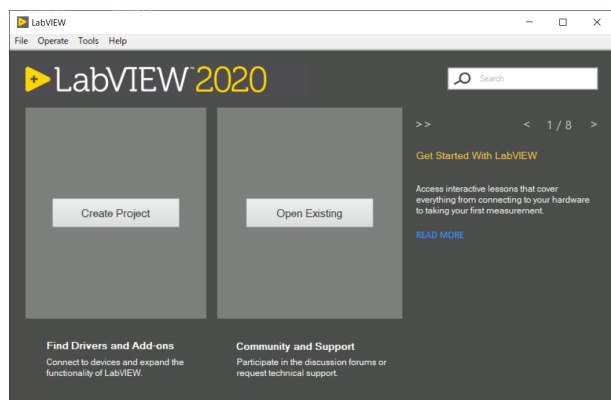


図 3-12 LabVIEWの初期画面

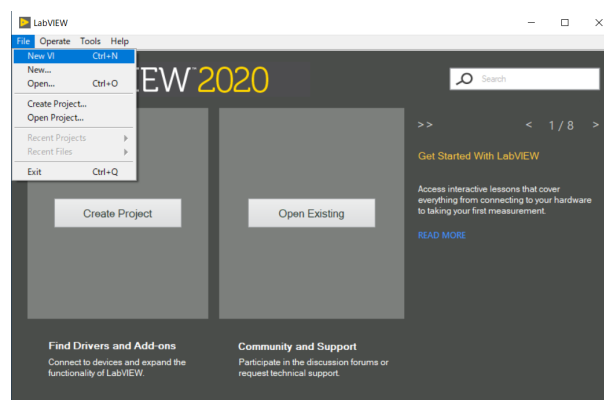


図 3-13 新規VIの作成

初めてVIを作るときには、**図3-13**のように画面の左上のFile》New VIまたは、キーボードの「CTRL」キーと「N」キーを同時に押します。LabVIEWではマウス操作だけではなく、キーボードをある組み合わせで押すことによって、素早く操作を行えるようになっています。

新規VIが作られましたね。改めて、画面を見てみましょう（**図3-14**）。

左にある背景が灰色のものが「フロントパネル」といって、ボタンを押してプログラムを操作したり、グラフを見て測定結果を確認したりするために使います。今は何もありませんが、ここにボタンやグラフを作っていくことで、そのような画面を作ることができます。右にある背景が白いものが「ブロックダイアグラム」です。プログラムを作る作業はこのブロックダイアグラムで行います。まずは、簡単な足し算や掛け算といった四則演算のプログラムを作ることで、基本的なLabVIEWの使い方を見ていきましょう。

まずフロントパネルに計算に必要な部品を4つ置いていきます。フロントパネルの上でマウスを右クリックしてみてください。「制御器パレット」と呼ばれるものが表示されます（**図3-15**）。これからの作業では、このパレットから必要な部品を選んでフロントパネルにおいていきます。最初からパレットが表示されているかもしれません。パレットを閉じても右クリックで現れます。

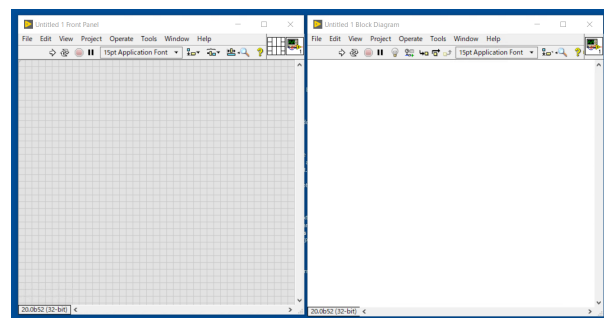


図 3-14 フロントパネルとブロックダイアグラム

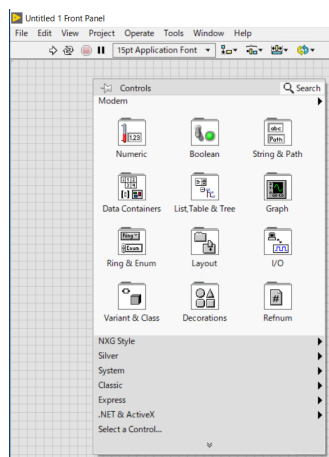


図 3-15 制御器パレット

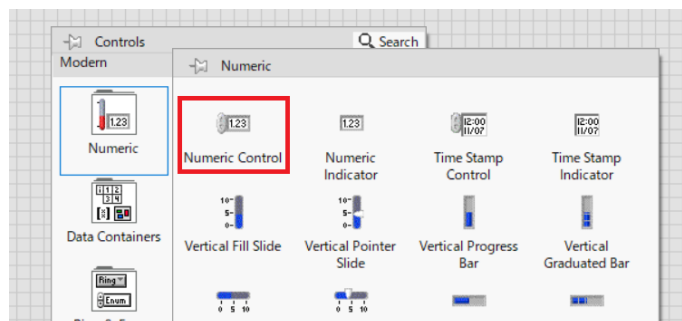


図 3-16 Numeric Control

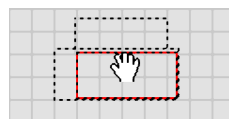


図 3-17 Numeric Control選択後

「Numeric」アイコンにマウスを合わせると、数字に関連したいろいろな部品が表示されます。ここで「Numeric Control」を左クリックしてみましょう（図3-16）。そうすると図3-17のように、Numeric Controlをもって動かすことができます。もう一度フロントパネル上で左クリックすると、フロントパネル上に置くことができます（図3-18）。LabVIEWでは、このように部品をパレットから選んで、フロントパネル上に置くことで画面を作っていきます。同じように、「Numeric Indicator」を置いてみてください（図3-19）。なんだか2つの見た目は違いますね。「Numeric Control」は実は「制御器」と呼ばれる部品グループの1つです。「Numeric Indicator」は「表示器」グループに所属し

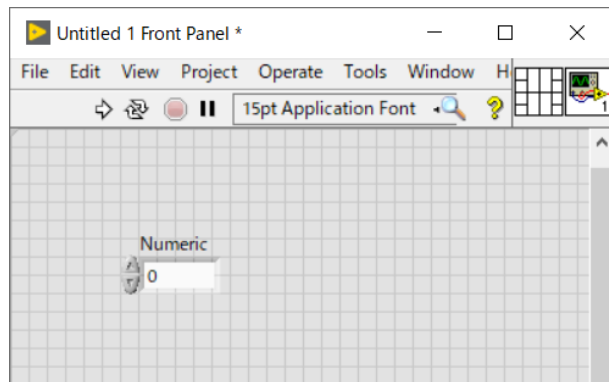


図 3-18 Numeric Control配置後

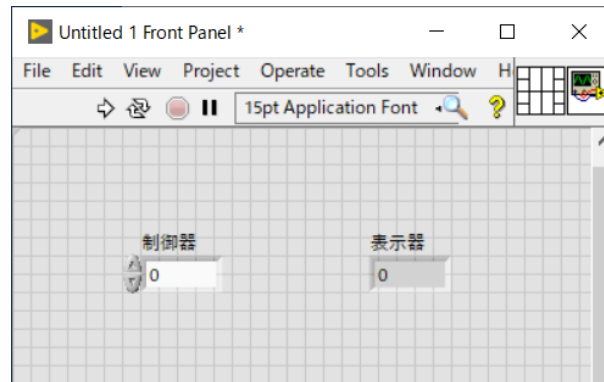


図 3-19 制御器と表示器

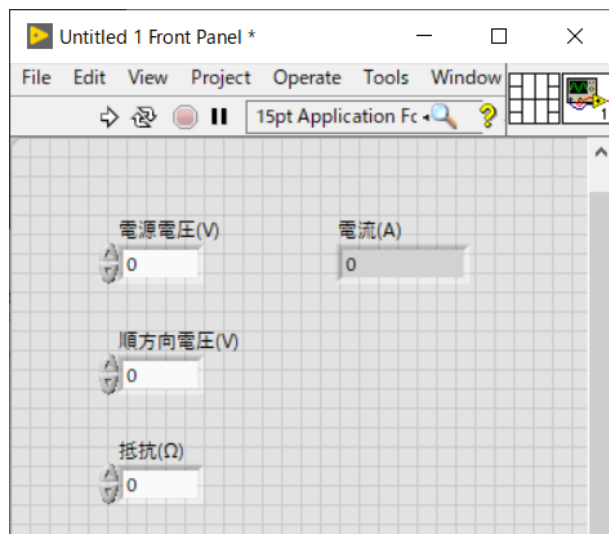


図 3-20 ここまでのフロントパネル

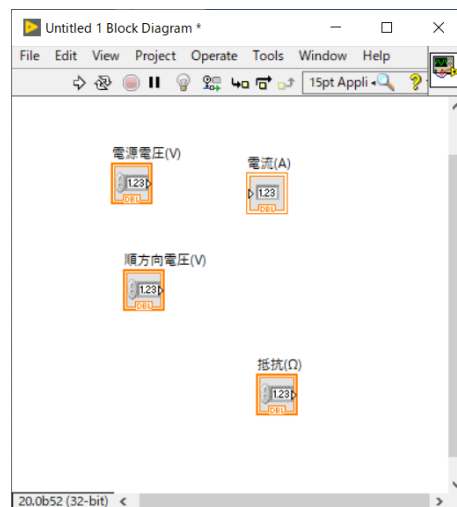


図 3-21 ここまでのブロックダイアグラム

ています。制御器を使えば、ボタンを押す、数を入力するなどの操作を行うことができます。一方で表示器では、計算結果を確認したりグラフを見るなどの、プログラムが行った作業の結果を確認するために使います。

このままの勢いで、あと2つ数値制御器を追加して、数値制御器が3つ、数値表示器が1つ準備された状態にしてください(図3-20、図3-21)。制御器や表示器の名前をダブルクリックすると、名前を変えられます。「Numeric Control」のままでは何をやるものかわからないので、図3-20のように名前を付けてあげてください。プログラミングの世界では、名前を正しくつけることが非常に大切です。名前を付けないと、どの部品が何のためにあるのかわかりません(図3-22)。わからないとプログラムを使えません。名前に情報を詰め込むことで、使える、かつ使いやすいプログラムを作ることができます。ここまででフロントパネルの作業は完了です。続いてブロックダイアグラムを見てみましょう。ちなみにフロントパネルとブロックダイアグラムの切り替えは、「CTRL」キーと「E」キーで簡単にできますので試してみてください。

ブロックダイアグラムでの作業は初めてですが、すでに何か置かれていますね。実はフロントパネルに制御器や表示器を置いたときに、これらのモノは自動的に作られていたのです。このブロックダイアグラムにあるモノを「端子」と呼びます。制御器/表示器と端子はそれぞれペアになっています。「電源電圧(V)」の数値制御器は、ブロックダイアグラムの「電源電圧(V)」端子と対応しています。フロントパネルで入力した値が、この端子から出力されます。実際に見てみましょう。

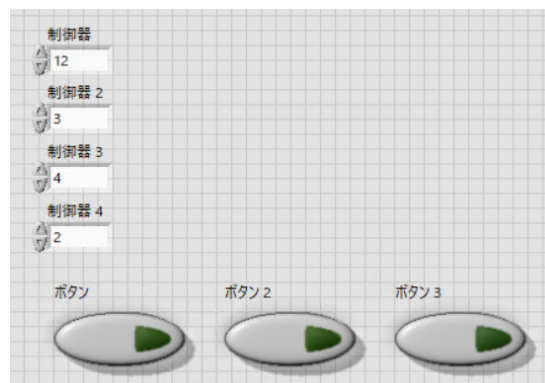


図 3-22 名前がないとよくわからない

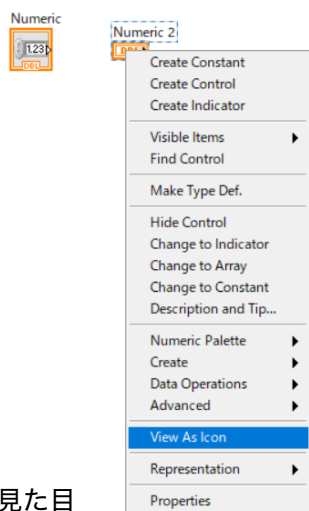


図 3-23 端子の見た目

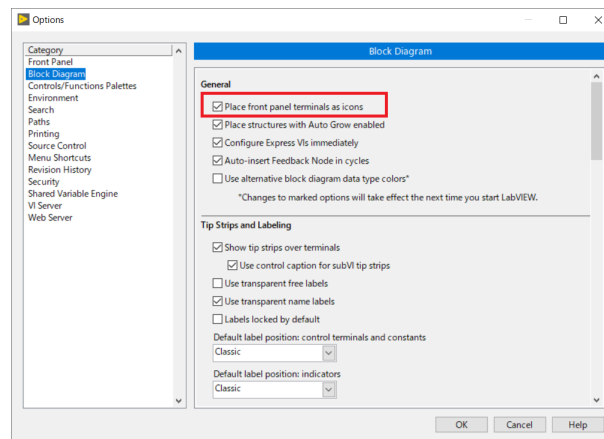


図 3-24 オプション画面

よく見ると、サンプルで使っている端子と練習で作った端子は見た目がなんだか違いますね（図3-23）。安心してください、どちらも同じもので、見た目が違うだけです。端子を右クリックして、「View As Icon」にチェックがなければ、ちょっと細長い見た目になります。常にこの見た目にしたいければ、LabVIEW上部のTools 》Optionsを選択してオプション画面（図3-24）を表示して、Block Diagramカテゴリの「Place front panel terminals as icons」のチェックを外してください。

制御器で値を設定して、表示器でその値を表示してみます。そのためには、「電源電圧(V)」制御器と「電流(A)」表示器を繋げてあげる必要があります。LabVIEWでは、ワイヤを使って互いを繋がめます。マウスカーソルを、制御器端子の右側の▽に合わせてみてください。カーソルが糸巻マークになります（図3-25）。これがワイヤを繋げるサインです。ここで左クリックすると、マウスの後を点線がついてきます（図3-26）。このようにワイヤを使ってLabVIEWでは、端子間でデータを受け渡します（図3-27）。

ここまで出来たら、フロントパネル上で「電源電圧(V)」制御器の値を好きな値にして、VIを実行してみてください。VIの実行は実行ボタンだけでなく、「CTRL」キーと「R」キーでもできます。制御器の値が、表示器に表示されたでしょうか。

このワイヤは練習のために使ったものなので、削除しましょう。先ほど作成したワイヤをクリックして、ワイヤが点線で囲まれたら、「BACK SPACE」または「Delete」キーを押すとワイヤが削除されます。

では今度は、関数を使ってプログラミングをしていきましょう。ブロックダイアグラムで右クリックしてください。フロントパネルの時と同じようにパレットが表示されますが、見た目が少し違いますね（図3-28）。これは関数パレットといって、ここからプログラミングのためのいろいろな関数を使うことができます。基本的な使い方はフロントパネルの時と大きく変わらないので安心してください。

今回は、電源電圧、順方向電圧、抵抗の大きさから、LEDに流れる電流の大きさを計算するプログラムを作ります。プロ

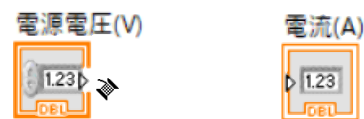


図 3-25 糸巻アイコン



図 3-26 配線中



図 3-27 配線完了

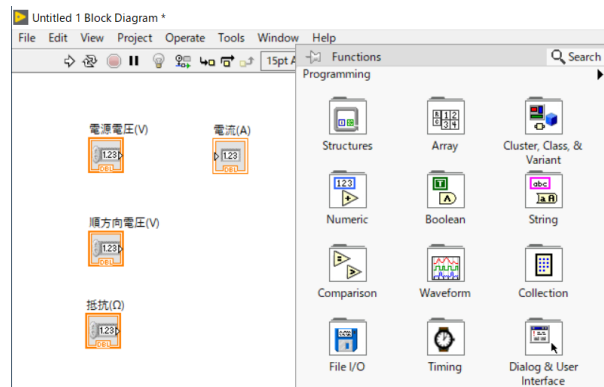


図 3-28 関数パレット

グラミングの前に計算式を考えましょう。

オームの法則から、電流は電圧÷抵抗ですね。電圧は電源電圧の大きさですが、順方向電圧を忘れてはいけません。実際に抵抗にかかる電圧は、電源電圧から順方向電圧を引いた値になります。順方向電圧は電流が変わってもあまり変化しない性質があるので一定と見なし、抵抗だけで計算します。

よって計算式は、

**抵抗とLEDに流れる電流 = ( 電源電圧 - 順方向電圧 ) / 抵抗値**

になります。この通りプログラムを作ってみましょう。

まずは引き算を行います。関数パレットを表示させて、Numericから**Subtract**を選んでください。**Subtract**は引き算の意味です。制御器の時と同じように、ブロックダイアグラムに置きます (図3-29)。

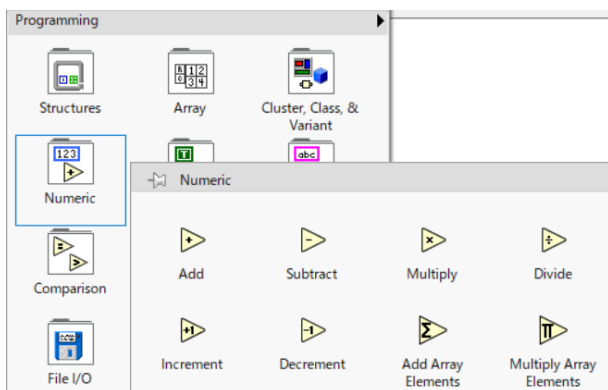


図 3-29 引き算関数

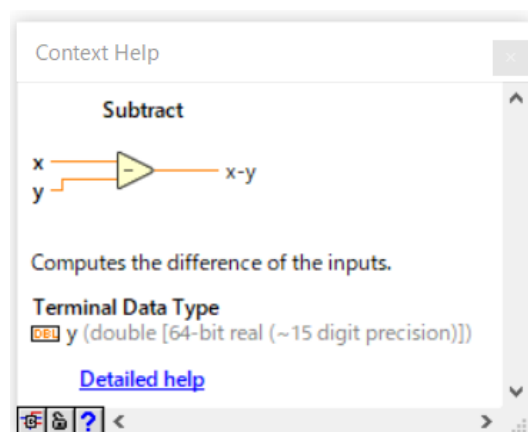


図 3-30 引き算関数のContext Help

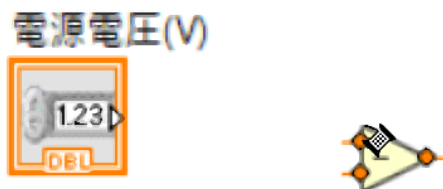


図 3-31 引き算関数への配線



使い方についてちょっと見てみましょう。「CTRL」キーと「H」キーを押してください。Context Help(コンテキストヘルプ)と呼ばれる、関数の使い方について教えてくれる画面が表示されます(図3-30)。もう一度このショートカットを押すと、画面は消えます。

どうやら左からxとyの値を入力すると、x-yの結果を右から出力してくれるようです。Context Helpでは関数の使い方の概略を確認することができます。もっと詳しい内容を見たければ、「Detailed Help」をクリックすると、もっと詳しい内容が書かれたヘルプを見ることができます。

ではこの引き算の関数の左上に「電源電圧(V)」制御器を、左下に「順方向電圧(V)」制御器を配線してみましょう。先ほどの制御器と表示器のワイヤと手順は同じです。引き算関数の上にマウスカースールを持っていくと、関数の上にオレンジ色の丸が表示されます。そこにマウスカースールを合わせると、カーソルが糸巻になり、ワイヤを作れるようになります(図3-31)。この状態で左クリック、そして「電源電圧(V)」制御器の右側にある▷の上で再度左クリックをして、ワイヤを作ります。ここまで出来たら、同じように「順方向電圧(V)」制御器の配線もしてください。図3-32のようにになります。最後に割り算をしましょう。これはDivide 関数です。関数パレットのSubtract 関数の2つとなりにあります。引き算の時と同じように配置と配線をしてください。図3-33のようにになります。フロントパネル(図3-34)で値を入力して実行すると、LEDに流れる電流値を確認することができます。



図 3-32 引き算関数への配線完了後

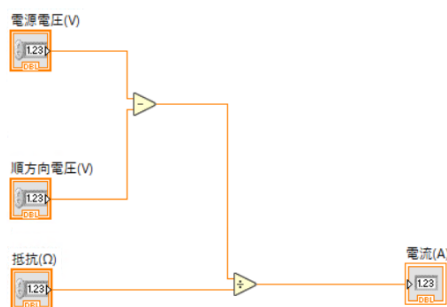


図 3-33 ブロックダイアグラム完成

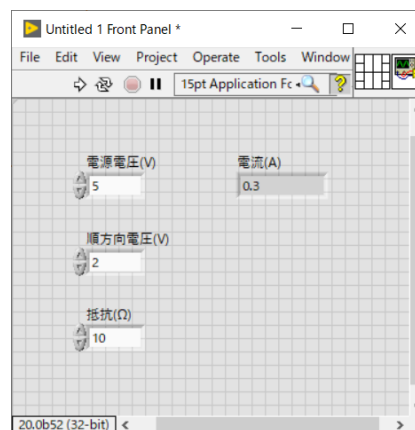


図 3-34 フロントパネル完成

ところで順方向電圧の値にマイナスの値を入れることができますが、これは少し変です。順方向電圧はLEDを光らせるために必要な最低限の電圧なので、これがマイナスということはありません。LabVIEWでは、入力できる数に制限をかけることができます。「順方向電圧(V)」制御器を右クリックして、右クリックメニューの一番下から「Properties」を選んでください。プロパティ画面が表示されたら、「Data Entry」を選択してください。「Use Default Limits」のチェックをクリックで消します。

「Minimum」を「0」に、「Response to value outside limits」を「Coerce(強制)」にします。これで順方向電圧の値は0が最小になって、これ以上小さい値を入力すると、強制的に0にされます。併せて便利のように、Incrementの値も、「0.1」にしておきましょう。OKボタンを押してプロパティ画面を閉じ、動作を確認してみてください。「0」より小さな値にはならず、制御器左側にある値を増やす、減らすボタンを押すと、値が0.1刻みで変化することが確認できます(図3-35)。

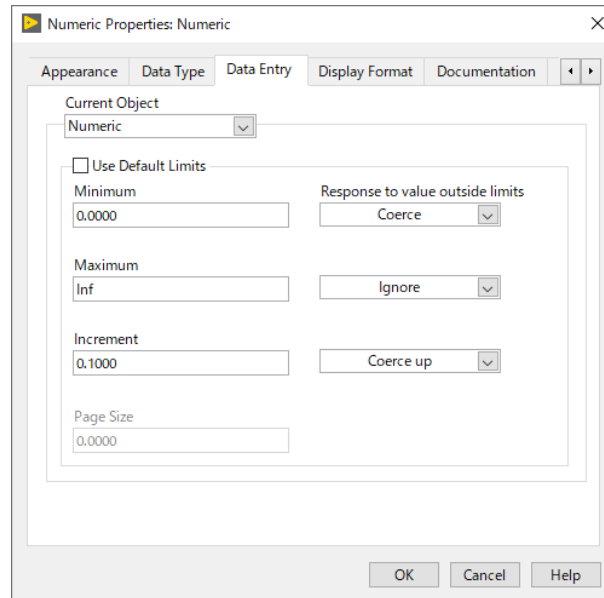


図 3-35 Numeric Controlのプロパティ

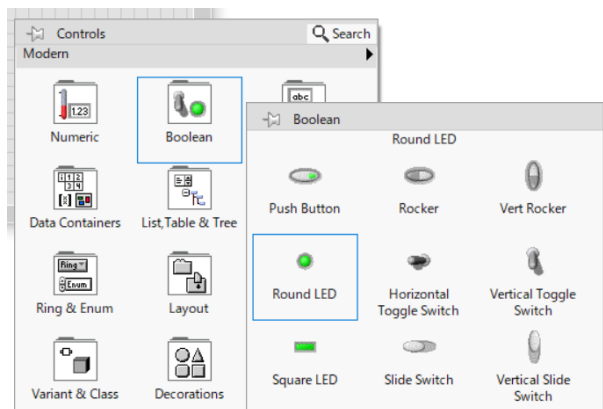


図 3-36 LED Indicator

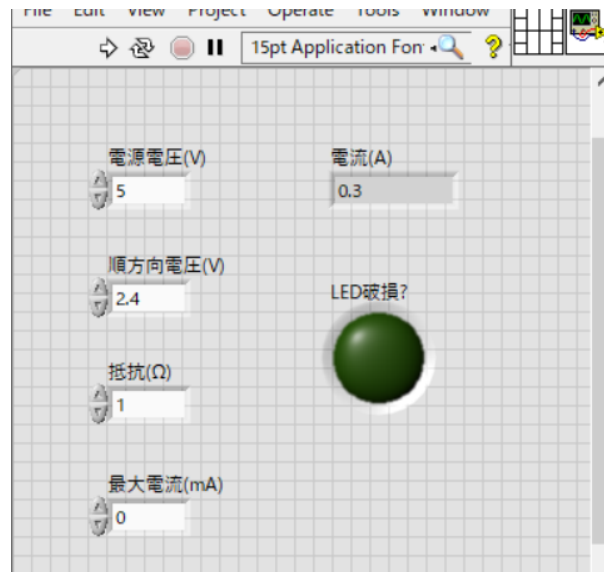


図 3-37 LED表示器の配置後

ブロックダイアグラムには直接現れない部分なので忘れがちですが、こういった「入力できる値を制限する」ことはとても大切です。使いやすくすることで、作ったプログラムに対する評判がよくなります。入力してほしい値を制限しておけば、プログラムが変な処理をすることもあります。プログラミングでは、こういった**ユーザビリティ**と呼ばれるプログラムの使いやすさを考えることも実は大切なのです。

ここまでのところで一度、VIを保存しましょう。LabVIEW上部のFile 》 Saveで保存ができます。または「CTRL」キーと「S」キーでも保存できます。好きな名前を付けて、VIを保存してください。

もう少し今のVIを進化させてみましょう。今度は、電流が指定した値より大きくなった場合、LEDが破損したことがわかるようにしてみます。数値制御器を1つ追加して、名前を「最大電流(mA)」とします。同じように、今度はBooleanパレット（図3-36）から、「Round LED」をフロントパネルに追加します。名前を「LED破損?」とします（図3-37）。

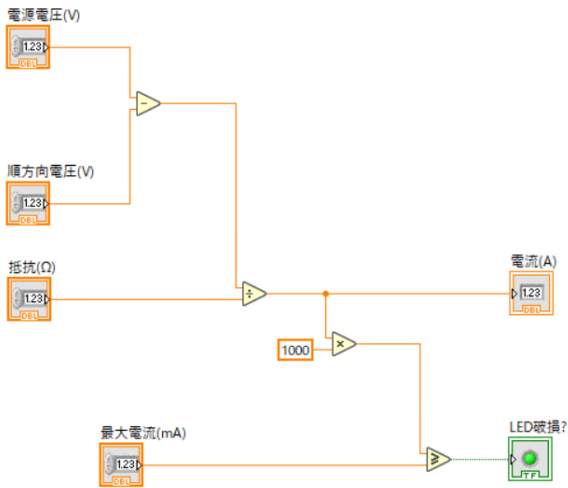


図 3-39 完成後のブロックダイアグラム

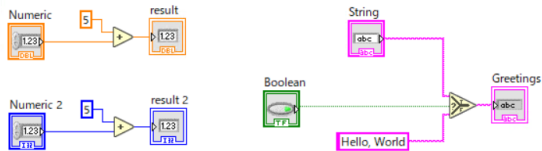


図 3-40 データタイプの例

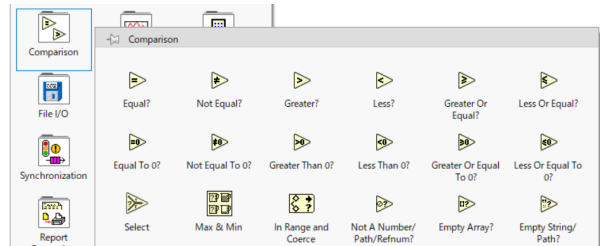


図 3-38 Comparison Palette

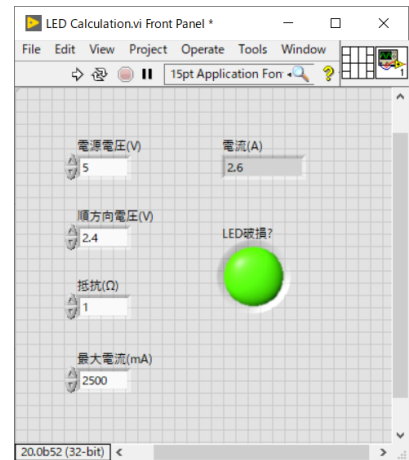


図 3-41 LED破損の結果

計算された電流と、「最大電流(mA)」制御器の値を比較します。関数パレットのNumericパレットの中から**Multiply** 関数と、Comparisonパレット (図3-38) の中にある、**Greater Or Equal?** 関数を選択してブロックダイアグラムに置きます。図3-39のように配線します。電流に1000の値を掛け合わせている理由はわかりますか？計算結果はアンペアですが、最大電流の入力はミリアンペアなので、単位をそろえるためにこの作業を入れています。

ここでのポイントは、「LED破損？」制御器とそれにつながるワイヤの色です。プログラミングでは、「どんなデータを取り扱うか」が大切です (図3-40)。これまで見てきたオレンジ色や青色の端子やワイヤは、数のデータを取り扱います。オレンジ色は小数データで「1.2345」などを表せます。青色は整数のみを表せます。緑色は「ブール」と呼ばれるデータで、F (False) かT (True) の2つの値しか取り扱いません。ピンク色の端子やワイヤは文字を取り扱って、「Hello, World」などの文字を受け渡しています。LabVIEWでは何のデータを扱うかを色で表現しているので、データと色の対応を覚えてしまえば、簡単に見分けることができます。

VIを実行して動作を確認してみましょう。最大電流を超えた電流が流れると、「LED破損？」表示器が明るくなって、破損したことがわかります (図3-41)。実際にはLEDが破損した場合、電流が流れないので電流の表示はゼロになってしまうところです。LED Simulator.viではそういった細かなところまで作りこんでありますので、今一度確認してみてください。

### 3.4 繰り返し実行されるようにする

もう少しプログラムを作りこんでみましょう。今は値を変更してその結果を確認するには、毎回VIを実行しなくてはいけませんでした。これは面倒です。VIは常に実行されていて、値を変更したらその結果がすぐに表示されるよう、VIを変更してみましょう。

プログラムの世界では、プログラムを繰り返し実行することを**ループ**と呼びます。ループでは、決まった作業を繰り返し実行して、停止命令を受け取ったら、そのループは停止します。LabVIEWでは、ループはStructuresパレットの中にあります (図3-42)。

では今までの計算作業をすべて、繰り返し実行されるようにしてみましょう。**Whileループ**をパレットから選択します。マウスカーソルが図3-43のように変わります。

そしてこのアイコンのまま、繰り返し処理をしてほしい箇所をループで囲みます。左上から右下へ、左クリックしたままマウスを移動させてみてください (図3-44)。そしてすべて囲ったところでマウスを離すと…？



図 3-43 ストラクチャ選択後のマウスカーソル

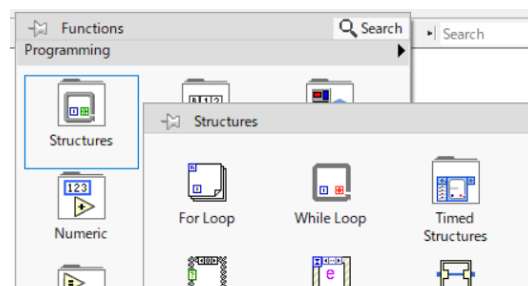


図 3-42 Structures Palette

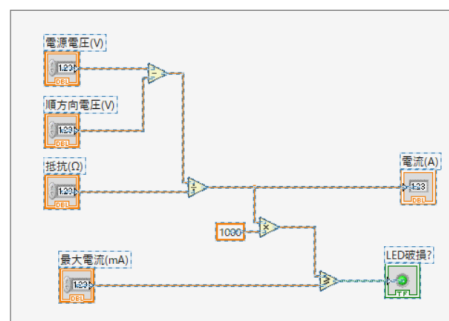


図 3-44 ループ作成中

Whileループが作られます（図3-45）。これで、ループで囲われた内側は、停止命令を受け取るまで繰り返し実行されます。「ではVIを実行してみましよう！」あれ、ちょっと待ってください。左上の実行ボタンがいつもの白色じゃないですね（図3-46）。

LabVIEWでは、プログラムに問題があって実行ができない場合、実行ボタンがこのように**壊れた状態**になります。先ほどまでは実行できたのに、何が問題でしょう…安心してください、LabVIEWでは例え実行ボタンが壊れたとしても、その

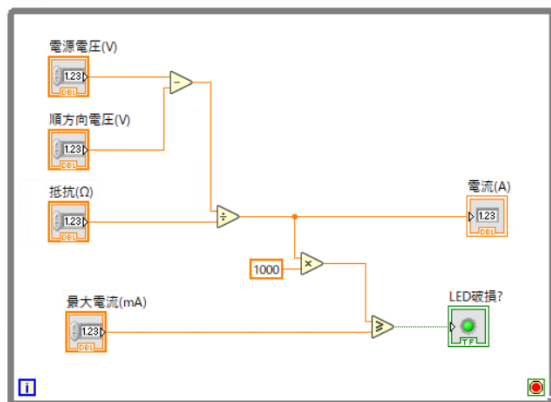


図 3-45 ループ完成

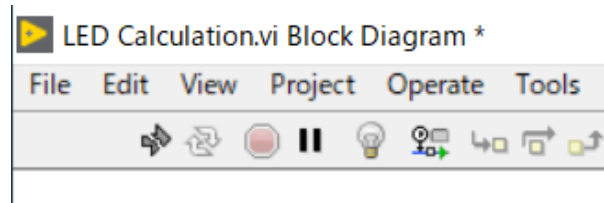


図 3-46 壊れた実行ボタン

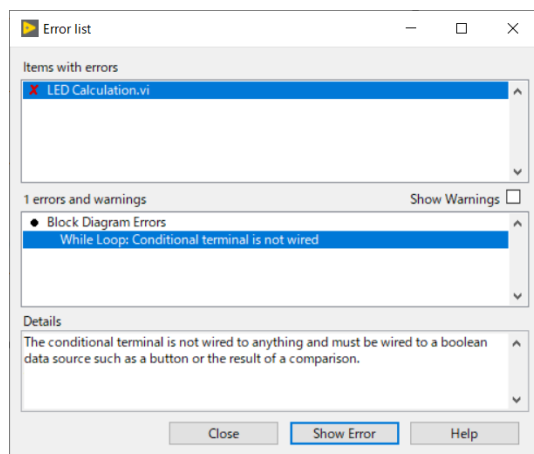


図 3-47 エラーのリスト

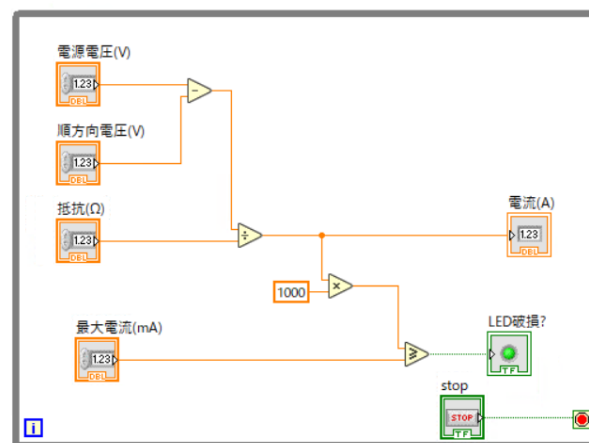


図 3-48 ここまでのブロックダイアグラム

原因がどこにあるか教えてくれるシステムが備わっています。実行ボタンは壊れていますが、無理やり実行してみましよう。実行ボタンを押すと、どこに問題があるかの**エラーリスト**が表示されます（図3-47）。今回は「While Loop: Conditional terminal is not wired」とあるので、Conditional terminalなるものにワイヤがつかないことが原因であるとわかりました。早速修正しましょう。エラーリストの**Show Error**ボタンでその場所にジャンプできます。

Conditional terminalは、Whileループ右下にある赤いアイコンです。このアイコンの上で右クリックして、「Create Control」を選んでください（図3-48）。そうすると、実行ボタンが直りましたね。実はWhileループでは、繰り返し処理を終わるために、ボタンなどのブールデータをこの端子に渡す必要があります。今の手順でブール制御器を作成すると、LabVIEWは自動で「stop(停止)」と書かれたボタン型のブール制御器を作成してくれます。VIを実行して、値の変更がすぐに反映されること、停止ボタンを押すとVIが止まることを確認して下さい。

ここまでで連続処理を作ることができましたが、連続処理を動かすうえで足りない処理が一つあります。実はWhileループを使って連続処理をすると、パソコンは全力でその処理を行おうとします。そうすると、パソコンの能力によってはマウス操作がカクカクしたり、最悪の場合他のソフトウェアが停止してしまったりします。これを避けるためにプログラミングでは、処理と処理との間に待ち時間を設けて、その待ち時間でパソコンが他の作業をできるようにします（図3-49）。

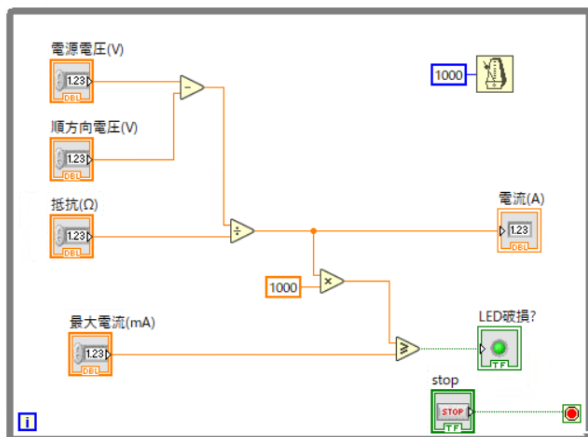


図 3-50 次のミリ秒倍数まで待機関数

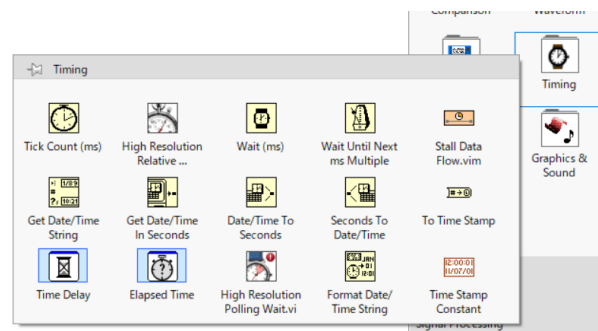


図 3-49 Timing Palette

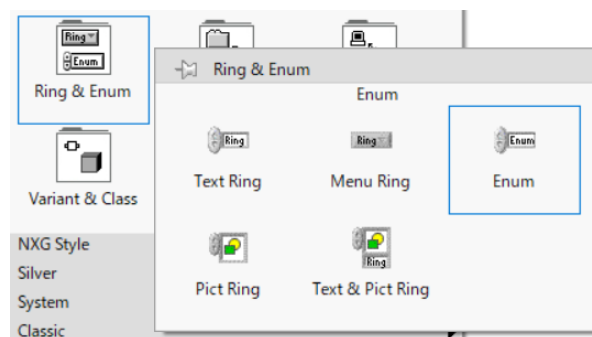


図 3-51 Ring & Enum Palette

タイミングパレットから、**Wait Until Next ms Multiple** 関数をWhileループの中に配置します。この関数は、処理と処理の間に待ち時間を入れて、パソコンの作業負荷を軽減します。「xx秒おきに測りたい」なんてときにも使える便利な関数です。これに定数を繋げます。定数とは、プログラムの実行中に絶対に変化しない値です。円周率や光速など、世の中で定義された値もすべて定数になります。定数を作るためには、関数の左側にマウスカーソルを合わせ、カーソルが糸巻になったら右クリック、Create Constantを選択します。ここで入力する値は「ミリ秒」なので、例えば一秒おきに処理をする場合には「1000」と入力します (図3-50)。

これで、1秒おきに計算をしてくれるプログラムが完成しました。

続いて電源電圧を、あらかじめ決められた値から選択するように変更しましょう。現状はどういった値でも電源電圧として設定できますが、実際には5/3.3/1.8Vのいずれかの電圧がよく使われます。今回はこの3種類から選べるようにしてみましょう。まずは「電源電圧(V)」制御器を削除します。フロントパネルまたはブロックダイアグラム上で制御器または端子をクリックして「BACK SPACE」または「Delete」キーで消します。端子を消すと、引き算の関数までつながっていたワイヤに×マークが付きましてね。これはワイヤが破損している状態で、正しく直さないとVIを実行できません。この時は壊れているワイヤを削除するか、「CTRL」キーと「B」キーのショートカットキーで、壊れたワイヤをいつぺんに消すことができます。ではフロントパネルにRing&Enumパレット (図3-51) 内からEnum 制御器を配置して、「電源電圧(V)」と名前を変更します。

Enumは**列挙体(れっきょたい)**といって、文字と数値のセットが複数まとめられたデータです。実際に作ってみるとよくわかります。列挙体制御器を右クリックしてプロパティを表示します。プロパティのEdit Itemsタブを選択します。ここで、列挙体のデータを定義します。Items列の"0"の隣をダブルクリックして「5V」と入力します。入力が終わったら、Enter

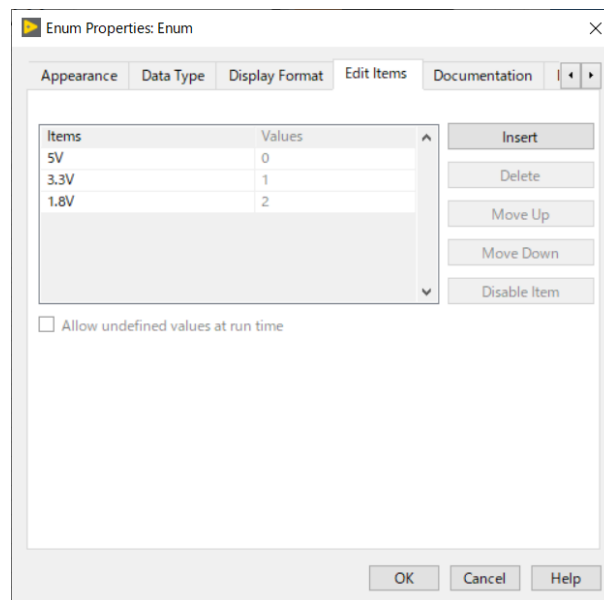


図 3-52 Enumの編集

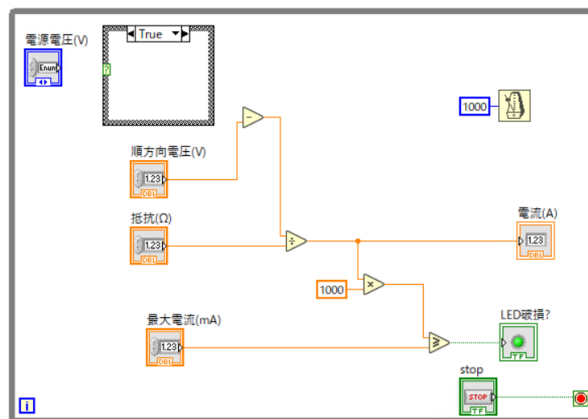


図 3-53 Case Structureの作成



キーを押して次の行へ進み、3.3Vと1.8Vの項目を追加します。それぞれの項目は数値と対応付けがされていて、5Vは0、3.3Vは1、1.8Vは2の値に対応しています。5Vを選ぶと0という値が、実際のプログラムでは計算に使われることを意味します。ここまで完了したらOKボタンを押してください（図3-52）。

さて電源電圧の値が、選ばれた電圧レベルと同じになるようにしなくてはなりません。Enumでは5Vを選択していても、実際にプログラムとして使われる値は「0」なので、このままでは「0V-順方向電圧」なんてことになります。ここで**ケースストラクチャ**を使用して、選ばれた電圧レベルと同じ値がプログラムで使われるように変更します。

ケースストラクチャは、場合分けの処理のために使います。例えば「ボタンが押されていていたらAの作業、押されていないかったらBの作業」のように、○○ならXX、XXなら○○のように状況によって行う処理を変えます。今回は、どの電圧が選ばれているかによって、引き算関数に渡す値を変更します。

ケースストラクチャは、Structuresパレットの中に**Case Structures**として準備されています。使い方はWhileループと同じで、パレットで選択し、作成したい場所に左上から右下へマウスをドラッグして作成します。今回は図3-53のように作ってください。

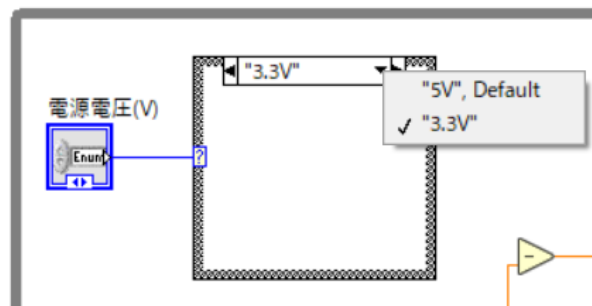


図 3-54 一部ケースが足りない

ケースストラクチャの左側には「？」マークの端子があって、ここにいろいろなデータを入力して、実行の条件を決めてあげます。ケースストラクチャを置いた最初は、ブールデータが入力されると思っているので、**True**か**False**の2種類のケースがあります。上部の◀▶を選択することで、ページを切り替えられます。ではこの「？」マークの端子に、先ほど作成したEnumを配線してみましょう。するとページに自動的に名前がつけましたね。▼マークを押してみると、5Vと3.3Vの2種類が表示されています。1.8Vのページが見当たりませんね（図3-54）。

ケースストラクチャの上部、電源電圧の名前が表示されているところで右クリックして、**Add Case for Every Value**を選択します。これですべての電源電圧パターンが表示されました（図3-55）。

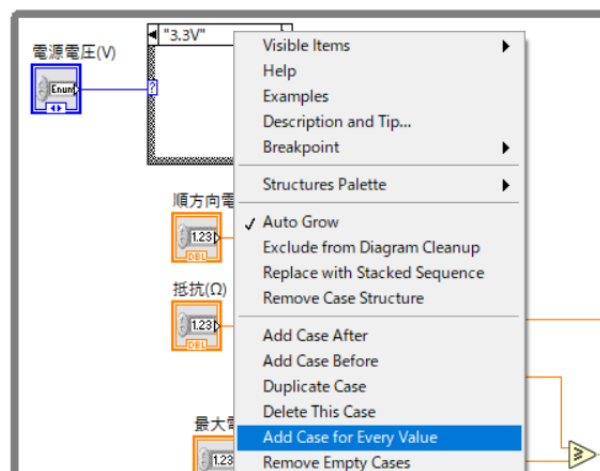


図 3-55 Add Case for Every Value

◀▶または▼マークから表示したいページを選ぶと、それぞ

れのページへ移動することができます。ページの中にプログラムを作成することで、その電源電圧値が選択されたときにどのような処理を行うかを定めることができます。手始めに5Vのページを表示して、Numericパレットの中からDBL Numeric Constantを置きます。この数値定数に「5」を入力します。数値定数と引き算の関数をワイヤでつなぎます（図3-56）。ケースストラクチャのふちがありますが、その辺はLabVIEWがよい感じにしてくれますので、気にせず今までと同じように配線をしてみてください。

配線が完了しました。ケースストラクチャのふちに四角が作られました。これは「トンネル」といって、ケースストラクチャやWhileループといった何かを囲うタイプの関数からワイヤを外に出すときに自動で作られるものです。先ほどは直接、数値定数から引き算関数にワイヤを持っていきましたが、ワイヤの作成途中でケースストラクチャのふちをクリックすると、自動でトンネルとそこまでのワイヤが作成されます。内側が白っぽいのは、他のページでどのような値を出すか決めていないことを教えてくれています。3.3Vと1.8Vのページで値を定義していませんでしたね。それぞれも数値定数を持ってきて、3.3と1.8の値を準備してあげましょう（図3-57）。すべてのページで値を定義すると、トンネルは塗りつぶされます（図3-58）。

最後に、フロントパネルのデフォルト値を設定しましょう。VIを保存して、一度閉じてください。再度VIを開くと、先ほど設定した各種値は元に戻ってしまっています。LabVIEWではデフォルト値という設定があって、VIを初めて開いたときの制御器や表示器の値を事前に設定しておくことができます。例えば順方向電圧を1.2V、抵抗を300Ω、最大電流を30mAに設定して、LabVIEW上部のEdit 》 Make Current

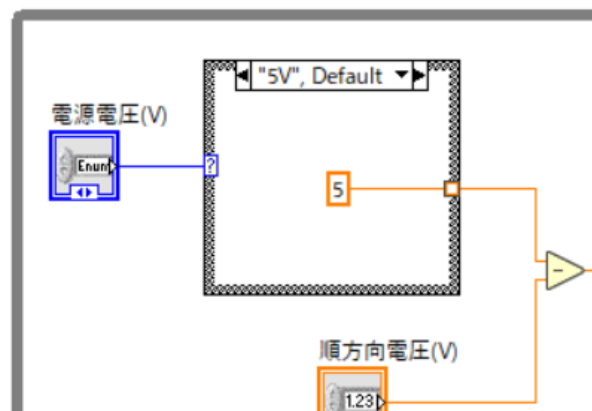


図 3-56 5Vケースの配線完了

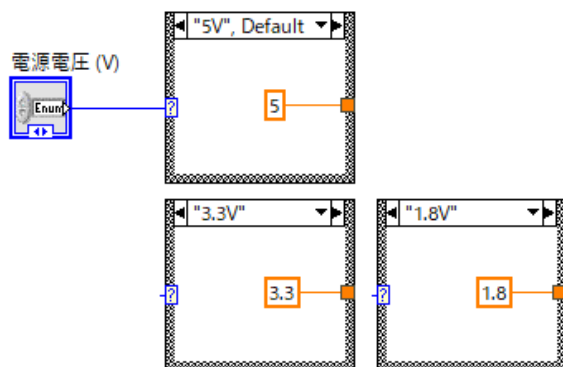


図 3-57 各ケースの内部処理

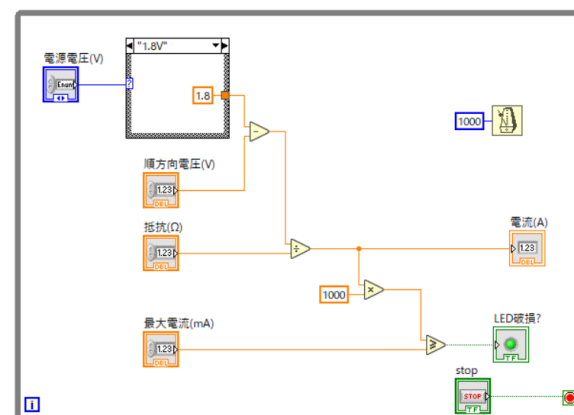
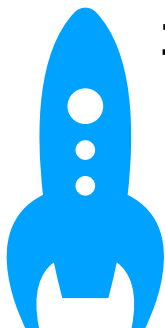


図 3-58最終的なVI

Values Defaultを選択します。VIを保存して再度VIを開くと、制御器の設定は先ほどの設定が保持された状態になっていることがわかります。

以上でVIの演習は終わりですが、以下の発展練習にも取り組んでみてください。

- LEDが破損した場合電流は流れないので、電流表示器の値は「0」にしたいです。ケースストラクチャではなく、Comparisonパレットの中にある**Select** 関数を使って、これを実現してみてください。
- LEDが壊れたということはとても大変です。「LED破損?」だけではなく、もっとインパクトのある方法で壊れたことを伝える方法はないでしょうか…Dialog & User Interfaceパレット内の**One Btn Dialog** 関数とケースストラクチャを組み合わせて、破損時にメッセージがポップアップするようにしてみてください。



## コラム3 LabVIEW NXGでのプログラミング

LabVIEW NXGでも制御器や表示器、関数やワイヤといった基本概念は変わりません。大きな

違いとしては、LabVIEWではフロントパネルとブロックダイアグラムは別々の画面でしたが、LabVIEW NXGでは一つの画面にまとめられている点です。それぞれパネル（図C3-1）とダイアグラム（図C3-2）と名前が変わって、スタイリッシュな見た目になっています。

LabVIEW NXGは表示言語の切り替えができるので、LabVIEW NXGを使えば日本語でプログラミングができます。

LabVIEW NXGを使うと、ウェブサイトも作ることができるので、これからのプログラミングはLabVIEW NXGがメインになってくるかもしれません。

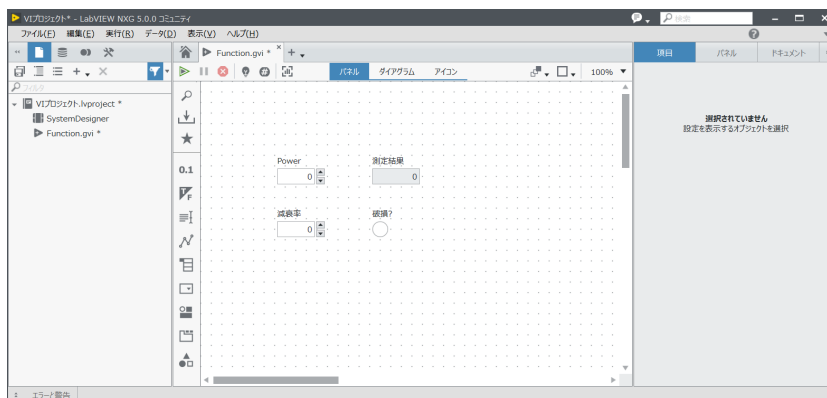


図 C3-1 LabVIEW NXGのパネル画面

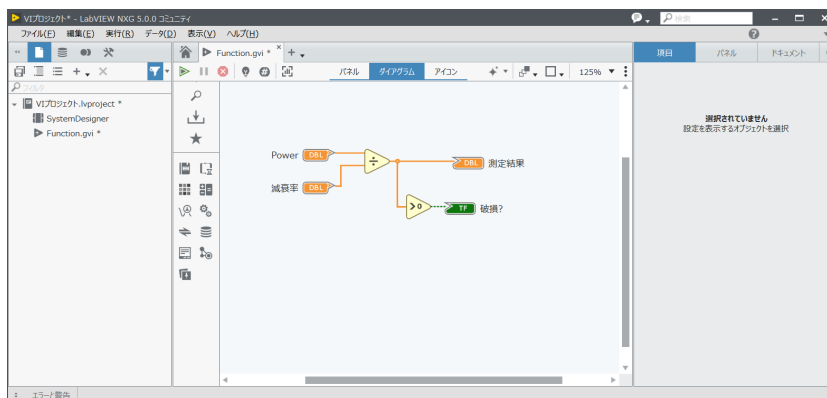


図 C3-2 LabVIEW NXGのダイアグラム画面

# 第4章

## 自分のアプリを作ってみる



LabVIEWで自分の趣味を深める・広げるアプリケーションを作りましょう。この章ではほとんどのPCに内蔵されている録音再生機能を使って実用性を感じられるアプリケーションの作り方を説明します。

**[キーワード]** スペクトログラム、サウンドコントロールパネル、ステレオミキサー、ステートマシン、状態遷移図、LabVIEWのサンプルVI、Finite Sound Input.vi、「Graphics & Sound」パレット、フラットシーケンスストラクチャ、ミリ秒待機関数、配列、波形データタイプ、1D配列反転関数、Forループ、シフトレジスタ、1次元配列、指標、2次元配列、強度グラフ、タイプ定義、Measurement File

### 4.1 趣味に活かすLabVIEWプログラミング

LabVIEWは技術者や科学者が仕事で使うために30年以上前に生まれました。Community Editionは仕事じゃなくて遊びに使うLabVIEWだよ、と急に言われてもすぐには何に使っていいのか分からないですね。

でも、LabVIEWのグラフィカルプログラミング言語を使うことで、これまで強く意識していなかったことでも、できるかもしれないということに気がついて、いつの間にか想像していたものよりもすごいものを作ってしまうかもしれません。ホームオートメーションや鉄道模型の制御などは、仕事で使われていたLabVIEWの延長上にありますのでプログラムが作れるようになりさえすればすぐに取り掛かることができますでしょう。

LabVIEWが長年磨いてきた計測や解析機能の優れた面を活かして多くの人に興味を持ってもらえるプログラムを考えてみました。PCとLabVIEWがあれば新しい機器を購入しなくても楽しめる**SoundVIEW** (図4-1) というお手製アプリケーションです。周波数成分の時間的な変化を色で表現する方法を**スペクトログラム**といい、ここでは横軸が時間で縦軸が周波数のグラフに、強さを黄色から赤そして黒までのグラデーションで表わしています。

ハイキングや山登りが好きな人は、山道を歩きながらどこで鳴いているのかわからない鳥の声を聞くことがあるでしょう。そして、声だけで鳴いている鳥の名前が分かったら素敵だなと感じる人は多いと思います。録音機材を持って野鳥の声を録音しに出かけるほど野鳥が好きな人たちが「鳴き声図鑑」をホームページで公開しています(図4-2)。図4-1はPCでホシガラスの鳴き声を聞きながら、SoundVIEWで周波数解析したものです。この方法で多くの野鳥の鳴き声を見ることができます。

外国語に興味のある人はYouTubeで見ることのできる、例えば高校生環境活動家グレタ・トゥンベリさんの国連気候行動サミットで行った演説を聞きながら発声のスペクトログラムを見ることができます。演歌が好きな人は八代亜紀のコブシをスペクトログラムで見たり、Jazzの好きな人はマイルス・デイビスのトランペットの音色を見たり、吉永小百合さんの朗読の息遣いを見たり、Webがまさに網のように皆さんの興味を捉えているでしょうからSoundVIEWで楽しむことがきっとあるはずです(図4-3)。

この章ではLabVIEWでアプリケーションを作る方法を説明したいと思います。始めにSoundVIEWのフロントパネルで、一連の操作を行ってみましょう。アプリケーション



図 4-2 鳴き声図鑑

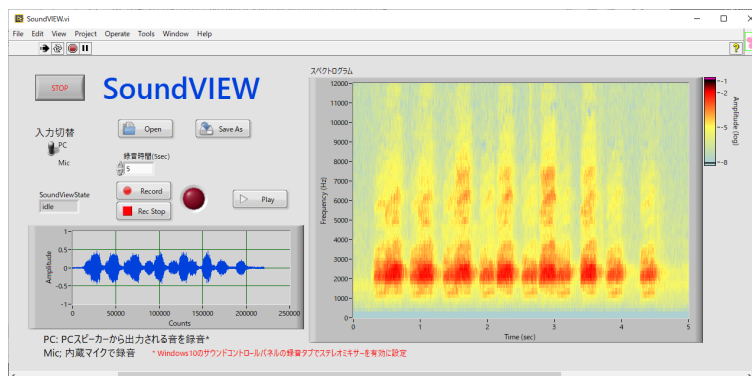


図 4-1 SoundVIEWのフロントパネル

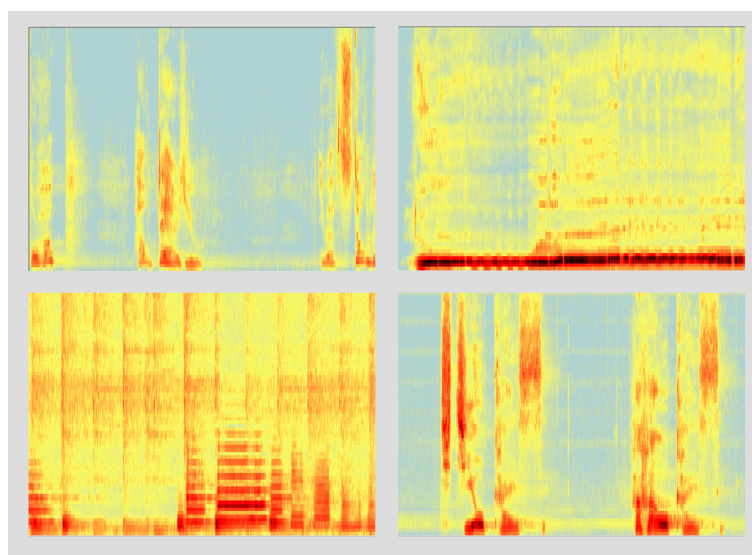


図 4-3 さまざまなスペクトログラム

プログラムなので、ボタン操作に応じた動作をするようになっています。プログラムのアーキテクチャ(構造)はステートマシンという考え方でできています。SoundVIEWのブロックダイアグラムを概観した後で、録音再生プログラムをステップを踏んで作っていきます。

## 4.2 SoundVIEWを操作してみる

第4章のプログラムフォルダーから**SoundVIEW.vi**を開いてください(図4-4)。SoundVIEWには録音機能があり、音源は「PC」と「Mic」を選択することができます。「PC」を選択した場合はWindows PCのステレオミキサー(図4-5)の機能でWebブラウザやMusic Playerなどから出力される音を録音できます。使っているPCによってはステレオミキサーが有効になっていない場合や、ステレオミキサーがサウンドコントロールパネルの録音タブに表示されていない場合もあります。その場合は「Windows10」と「ステレオミキサー」をキーワードに対処方法を検索してください。詳しく説明しているサイトが見つかるはずです。ステレオミキサーは再生リダイレクトやWhat U Hearなどと表示されていることもあるようです。

LabVIEWのサウンド機能では「既定のデバイス」がID=0、「既定の通信デバイス」があればID=1、残りの有効なデバイスに上から順にID=2以降が割り振られます。図4-5の場合は3つのデバイスが表示されていますが、一番上のデバイスは、外部マイクが接続されていないため有効ではなく、マイクが「0」、ステレオミキサーが「1」となります。外部マイクを接続して、有効にしている場合は外部マイクが「0」、マイクが「1」、ステレオミキサーが「2」となります。この



図 4-4 入力切替スイッチで音源を選択

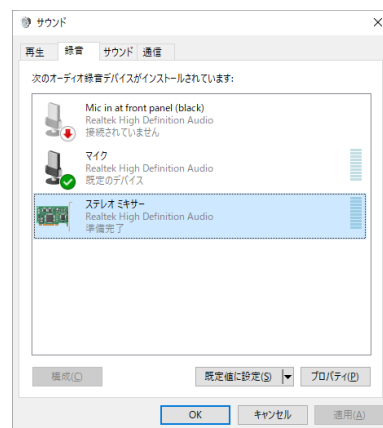


図 4-5 サウンドコントロールパネル



ようにデバイスIDはお使いのPCの状態により変わりますので、ブロックダイアグラムのrecStartサブダイアグラム（図4-6）で適切なID番号に変更してください（図4-7）。

第4章のプログラムフォルダの**AvailableDevice.vi**を開いて実行するとLabVIEWが認識している音源が表示されます（図4-8）。上欄が入力デバイスで下欄が出力デバイスですが、録音機能に使う入力デバイスは「マイク」（ID=0）とステレオミキサー（ID=1）になっています。

**AvailableDevice.vi**を開いている状態でコントロールパネルで設定を変更した場合は、

**AvailableDevice.vi**を一度閉じてから、再び開いて実行しなさいと反映されないので注意してください。

「Record」ボタンを押すと録音が始まります。録音時間は5秒が標準ですが、PCの能力が高ければ大きな値に変更しても動作するかもしれません。「Rec Stop」ボタンを押すと録音を停止します。録音中の波形は「Rec Stop」ボタンの下のグラフに表示されます。「Play」ボタンを押すとスペクトログラムが表示されて録音した音声再生されます。「Save As」ボタンを押すとファイルダイアログが表示されてファ

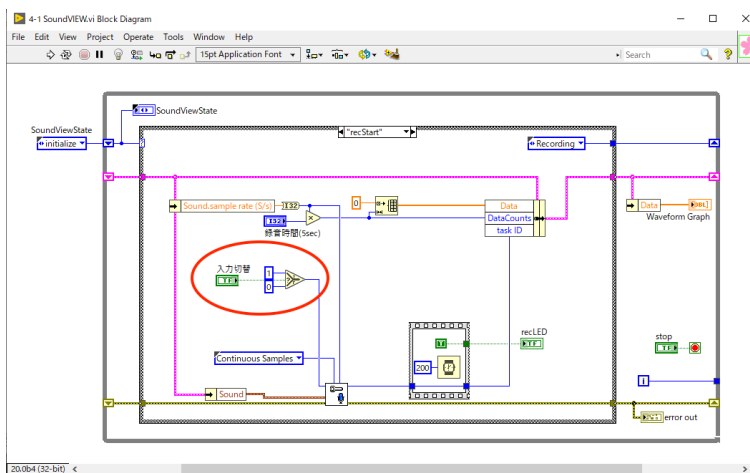


図 4-6 recStartサブダイアグラム

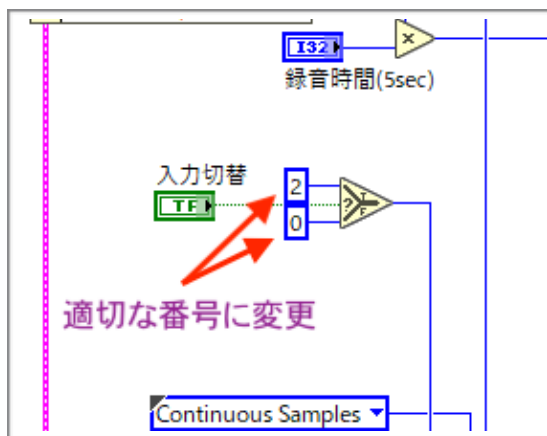


図 4-7 音源のID

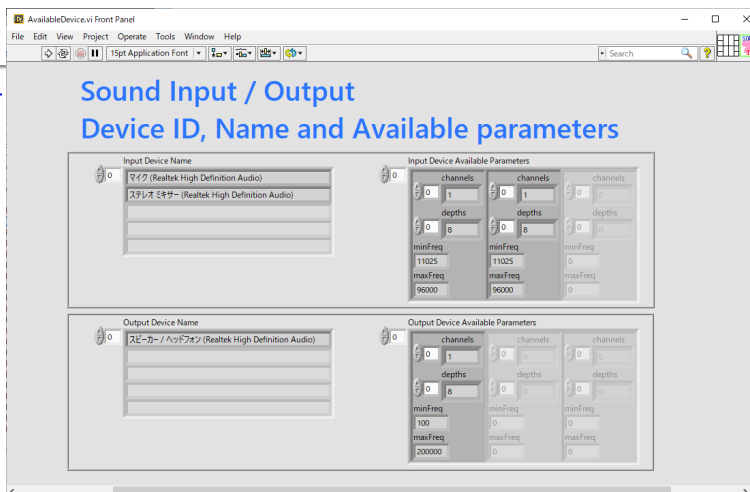


図 4-8 AvailableDevice.viを使った音源の確認



イル名を入力して保存することができます。「Open」ボタンで記録していたファイルを読み込むことができます。「STOP」ボタンが押されれば停止します。**SoundVIEWState**には現在の状態が表示されます。機能を確認しながら操作してみてください。

ブロックダイアグラムを見ると、大きなWhileループの中にケースストラクチャがあります(図4-9)。ケースストラクチャには8枚のサブダイアグラムがあります。ユーザーの操作やプログラムの流れに応じてケースセクタに入力される列挙体の値が変わり、サブダイアグラムが選択的に実行されています。このようなプログラムの書き方を**ステートマシン**と言います。

図4-10にプログラムの動作を模式的に描いた**状態遷移図**を示しました。

VIの**実行ボタン**を押されると**初期化**

(initialize) して**入力待ち** (idle) になります。**入力待ち** (idle) では**Record**ボタン、**Play**ボタン、**Open**ボタン、**Save As**ボタンが押されるのを待ちます。

**Record**ボタンが押されると**録音開始**

(recStart) 処理が行われて、**リングバッファに録音** (Recording) サブダイアグラムが**Rec Stop**ボタンが押されるまで継続します。

**Rec Stop**ボタンが押されると**録音停止**

(recStop) 処理を行って、**入力待ち** (idle) になります。

**再生** (Play)、**保存** (Save)、**読み込み**

(Load) は、対応するボタンが押されると処理を行い、**入力待ち** (idle) に戻ります。

実現したい機能をサブダイアグラムに小分けして、自然な操作ができる状態遷移を考えることで、複雑な動作のプログラムも作ることができます。

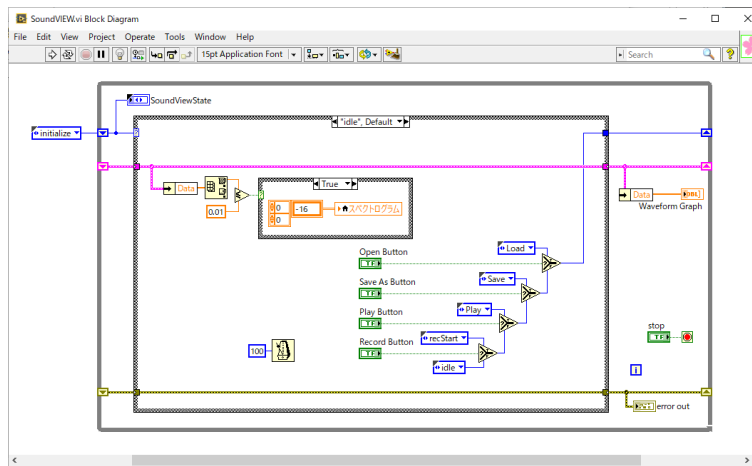


図 4-9 idleサブダイアグラムでの状態選択

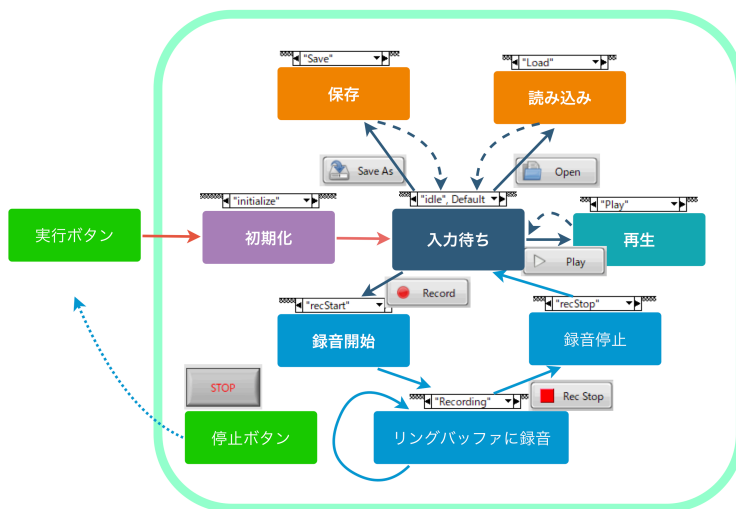


図 4-10 SoundVIEWの状態遷移図

SoundVIEWのスペクトログラムは「Play」サブダイアグラム（図4-11）にある **sub\_Spectrogram.vi** で作成しています。アイコンをダブルクリックしてVIを開き、ブロックダイアグラムを見ると図4-12のようになります。

**sub\_Spectrogram.vi** では入力である「data」から周波数解析パレットの関数を使ってスペクトログラムを得て2次元配列表示器「STFT Spectrogram」を出力します。表示するグラフのスケールや表示色の設定も表示器で出力しています。**sub\_Spectrogram.vi** のようにひとまとまりになる機能に inputs と outputs を決めて関数としたものを **サブVI** と呼んでいます。ブロックダイアグラムの見通しが良くなってプログラムの誤りを見つけやすくなる効果があります。

**sub\_Spectrogram.vi** の内部のプログラムの細かなことに興味がない人は、入力データを接続すればスペクトログラム

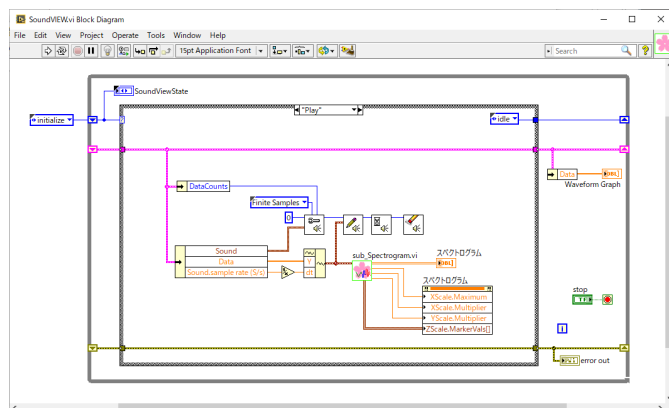


図 4-11 playサブダイアグラム

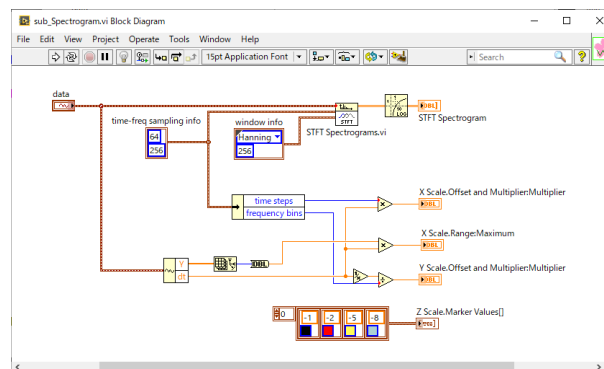


図 4-12 sub\_Spectrogram.viのブロックダイアグラム

が出力される関数として細部に立ち入らなくて済みます。プログラムを作る場合にも部分的に動作確認できるので完成度を高めることができます。サブVIは他のプログラミング言語ではサブルーチンに相当しますが、多くのプログラミング言語ではサブルーチンを単独で動作させることはできません。単体でサブVIの動作確認ができるのはLabVIEWの利点の一つです。

## 4.3 録音と再生のサンプルプログラム

どんなテーマであれLabVIEWで新しいことを始めるときにはサンプルプログラムを見るようにしましょう。たいていの場合良いスタートポイントを与えてくれます。

図4-13のようにヘルプメニューから「Find Examples...」を選択すると「NI Example Finder」が開きます。

「Directory Structure」で表示するボタンを押します。図4-14のように「Graphics and Sound」フォルダの「Sound」フォルダを開いて**Finite Sound Input.vi**を選択します。

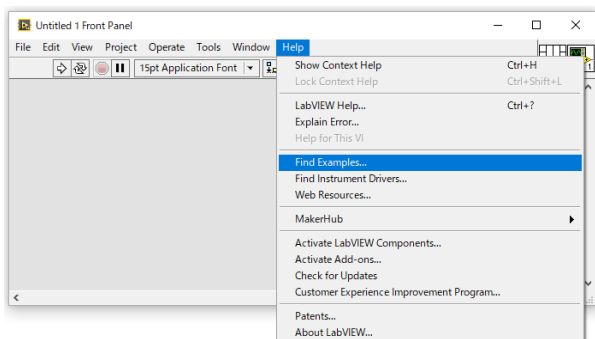


図 4-13 Find Examples...

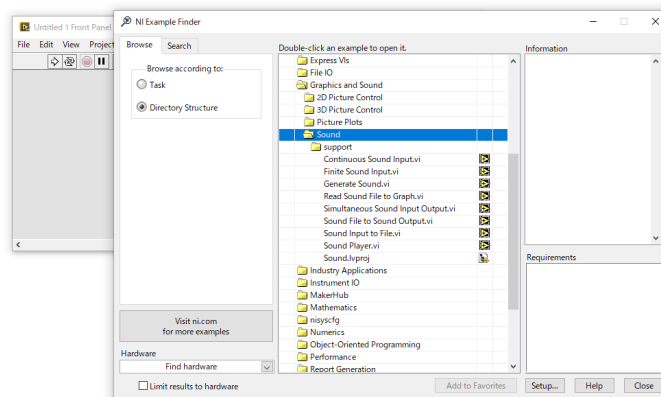


図 4-14 NI Example FinderのSoundフォルダ

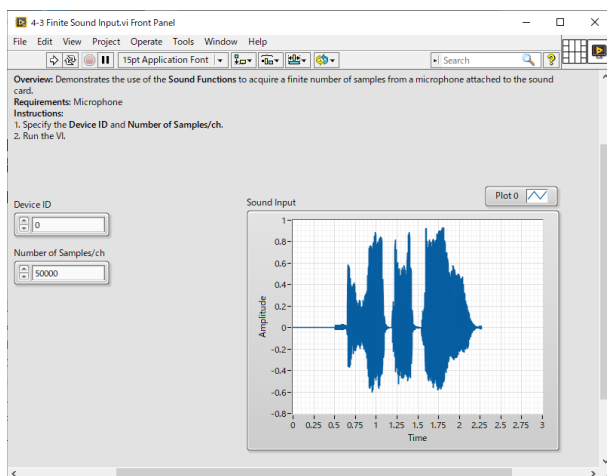


図 4-15 Finite Sound Input.vi

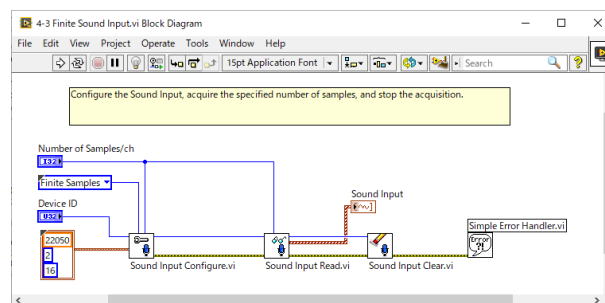


図 4-16 Finite Sound Input.viの  
ブロックダイアグラム

**Finite Sound Input.vi**が開きますので、自分の作業フォルダに保存します。「Device ID」に「0」、「Number of Samples/ch」に「50000」を入力して、マイクに向かって音を出しながら実行ボタンを押してください。図4-15のように2.2秒ぐらい音声データがグラフに表示されたと思います。データの先頭から約0.5秒までは録音されたデータではないので注意してください。ブロックダイアグラム（図4-16）を見ると、サンプリング周波数が22050Hz、チャンネル数が2、精度が16ビットで録音していることがわかります。アイコンの絵を見ると、スパナマーク→メガネ→消しゴムで、(1) 録音設定から(2)録音して(3)設定クリアという手順で進めていることがわかります。

それでは、このサンプルプログラムを改造して、スピーカーからおうむ返しに音を出すプログラムを作りましょう。

まず、**Finite Sound Input.vi**を**parrot.vi**という名前で保存してください。

サウンド関連の関数は関数パレットの「Graphics & Sound」パレットの「Sound」にあります（図4-17）。さらに「Output」を選択すると図4-18の音声出力の関数が出てきます。この中からスピーカアイコン、鉛筆アイコン、砂時計アイコン、消しゴムアイコンを**parrot.vi**のブロックダイアグラムにドラッグドロップします。砂時計アイコン**Sound Output Wait.vi**は、データが全て出力されるまで待つ関数です。図4-19のように配置して配線します。マイクで録音した音がすぐ再生されると思います。

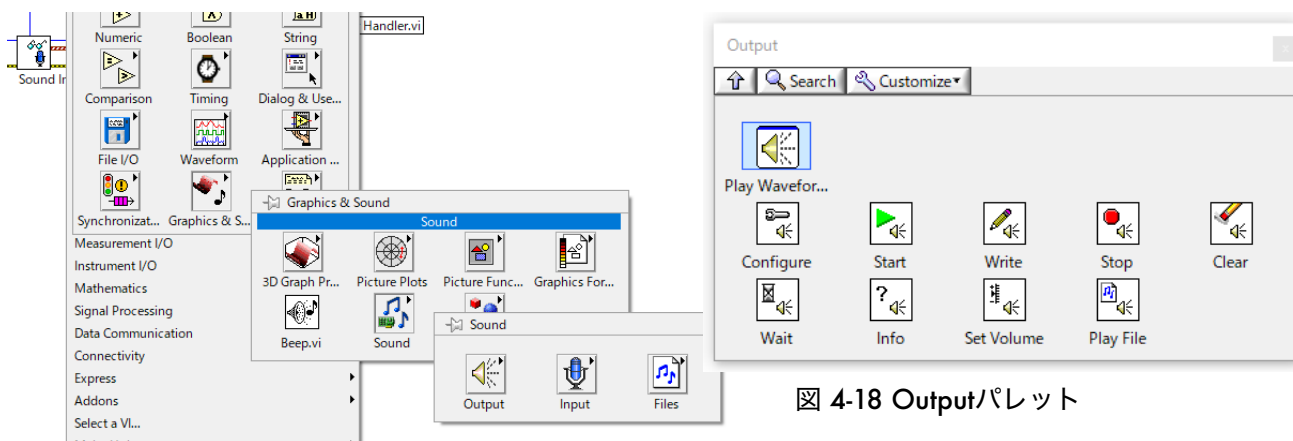


図 4-18 Outputパレット

図 4-17 Graphics & Soundパレット

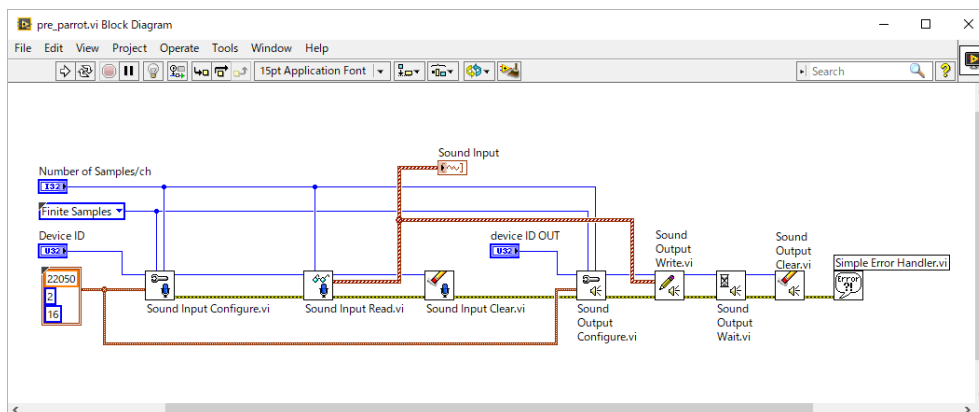


図 4-19 音声出力関数の  
配置と配線

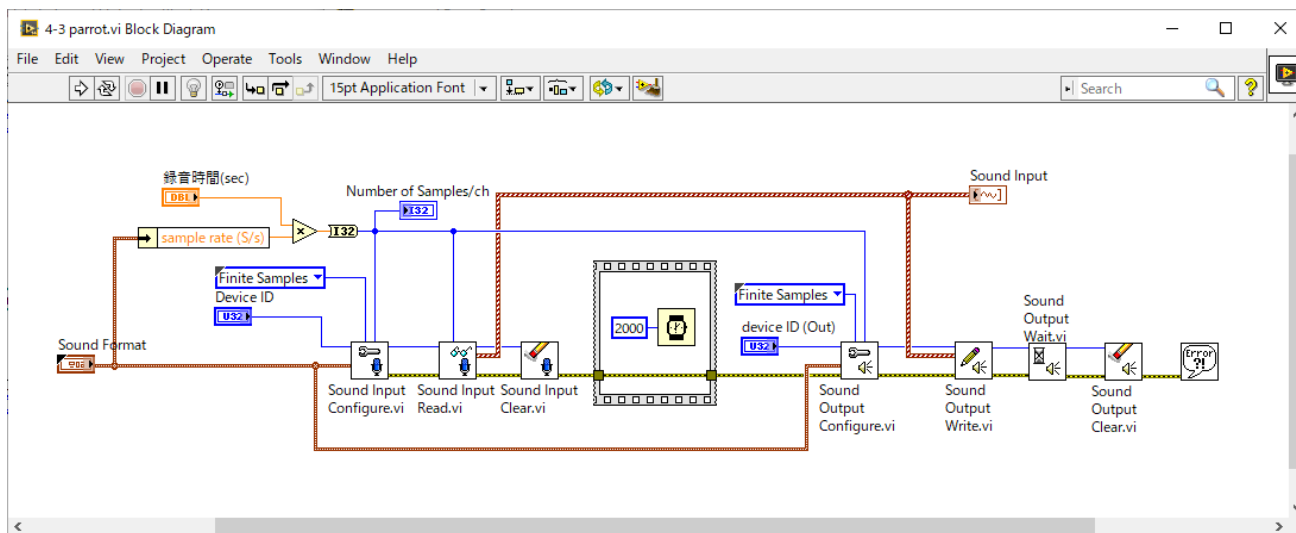


図 4-20 parrot.viのブロックダイアグラム

数秒の間を作るためにフラットシーケンスストラクチャを作り中に**ミリ秒待機関数**で2秒間待機させます。収録が終わり、数秒後に再生するオウム返しのプログラムができました。サンプル数で指定するよりも録音時間で指定する方が使いやすいと思いますので、その点も変更しました（図4-20）。

## 4.4 波形データと配列

parrot.viをreverse.viという名前で保存してください。音声を逆送りで再生するプログラムを作りましょう。先ほどまで使っていた録音データは茶色の太い線で表されていて、マイクで受けた音声データをスピーカー出力に配線することでおうむ返しできていました。ここで扱っていた録音データは左チャンネルと右チャンネルの音声データをひとまとまりにしたものです。サウンドフォーマットでチャンネル数を2に設定していたのでステレオデータになっています。それぞれのチャンネルのデータを見るためには**指標配列 (Index Array)** 関数を使い

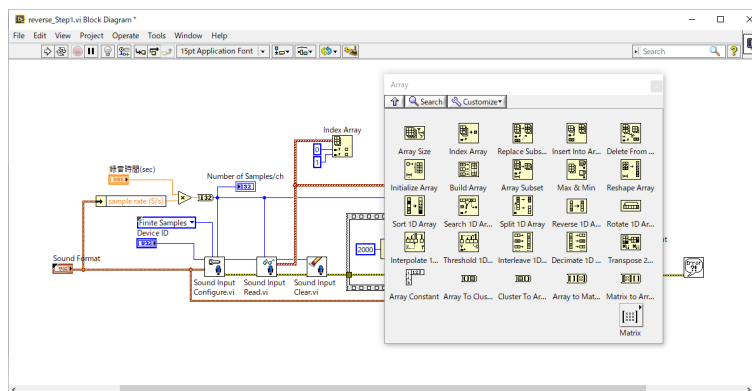


図 4-21 指標配列 (Index Array) 関数

ます。同じ種類のデータタイプをひとまとまりにして扱いやすくしたものを**配列 (Array)** と言います。配列にまとめられた個々のデータを**要素 (Element)** といい、指標という0から連続する番号で指定します。左チャンネルは指標0の要素、右チャンネルは指標1の要素です。例えば右チャンネルのデータは指標1の要素なので、右チャンネルのデータを扱うときには**指標配列 (Index Array)** 関数を使って指標1の要素を取り出します。配列パレットから**指標配列**関数をドラッグドロップします。アイコンの下部をクリックしてドラッグすると端子を増やすことができます。左側に取り出した要素の指標を入力します。ここでは0と1です(図4-21)。指標配列関数の右側の端子をワイヤリングツールで右クリックして「Create」から「Indicator」を選択します(図4-22)。波形表示器が作られます。要素に分解された後の配線は細くなっていることに注意しておいてください。フロントパネルの波形表示器は図4-23のようにt0、dt、Yで構成されています。**reverse.vi**の実行ボタンを押すと波形表示器にデータが表示されます。t0は録音の開始時刻、dtはサンプリング間隔を秒で表示しています。45 $\mu$ 秒間隔というのはサウンドフォーマットで指定したサンプルレート22050(S/s)の逆

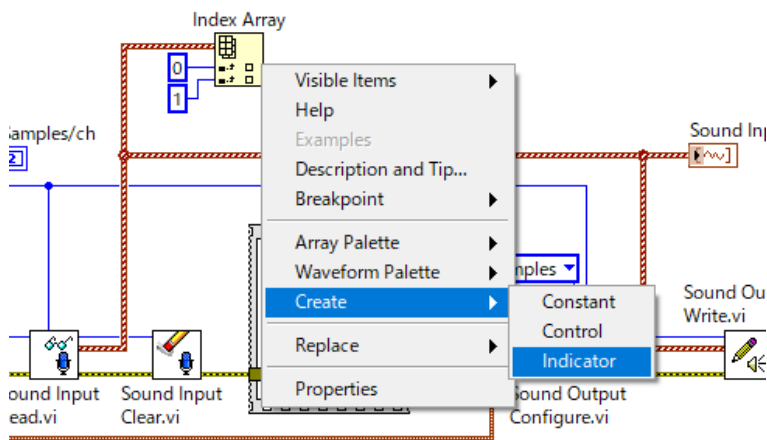


図 4-22 波形表示器の生成

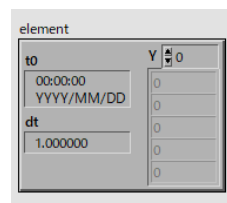


図 4-23 生成された波形表示器

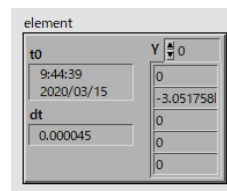


図 4-24 実行後の波形表示器

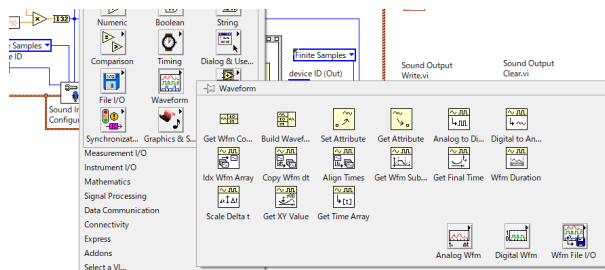


図 4-25 波形 (Waveform) パレット

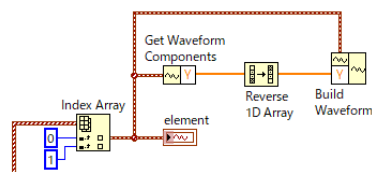


図 4-26 データ配列の前後反転

数になっています。Yにも値が入りました（図4-24）。Yが録音した音声データの配列です。Yの要素数はサンプルレートに録音時間（秒）をかけた数値になっています。Yと書かれている部分の右側が指標表示で、数値を増減させると表示される先頭の指標が変化します。波形データタイプは先頭の時刻t0とデータ間の時間差dtが分かるので、すべての要素のサンプリングされた時刻を知ることができます。一定の間隔でサンプリングされたデータを記録する方法として便利なので、波形データタイプを扱う関数を集めた波形（Waveform）パレット（図4-25）が用意されています。波形データタイプはサンプリング間隔とデータ配列がセットになっているため、周波数解析の関数の入力として便利に使われます。波形パレットの**GetWaveform Components**と**Build Waveform**をドラッグドロップしてください。配列パレットから「Reverse 1D Array」をドラッグドロップして、それらを図4-26のように配置して配線してください。このようにすることで波形データの配列データの前後を反転することができます。指標1についても同様に行います。配列パレットの「Build Array」を使って前後を反転させた2個の波形データを配列にして**Sound Output Write.vi**に入力します（図4-27）。**Build Array**も上下に伸縮して端子数を変更できるのでアイコンの見た目が変わります。できあがった**reverse.vi**を実行してみると遠い外国の言葉のようです。言葉をローマ字で書いてアルファベットを逆に並べ換え、それをローマ字読みして録音再生してみてください。なんとなく元の言葉に聞こえます。

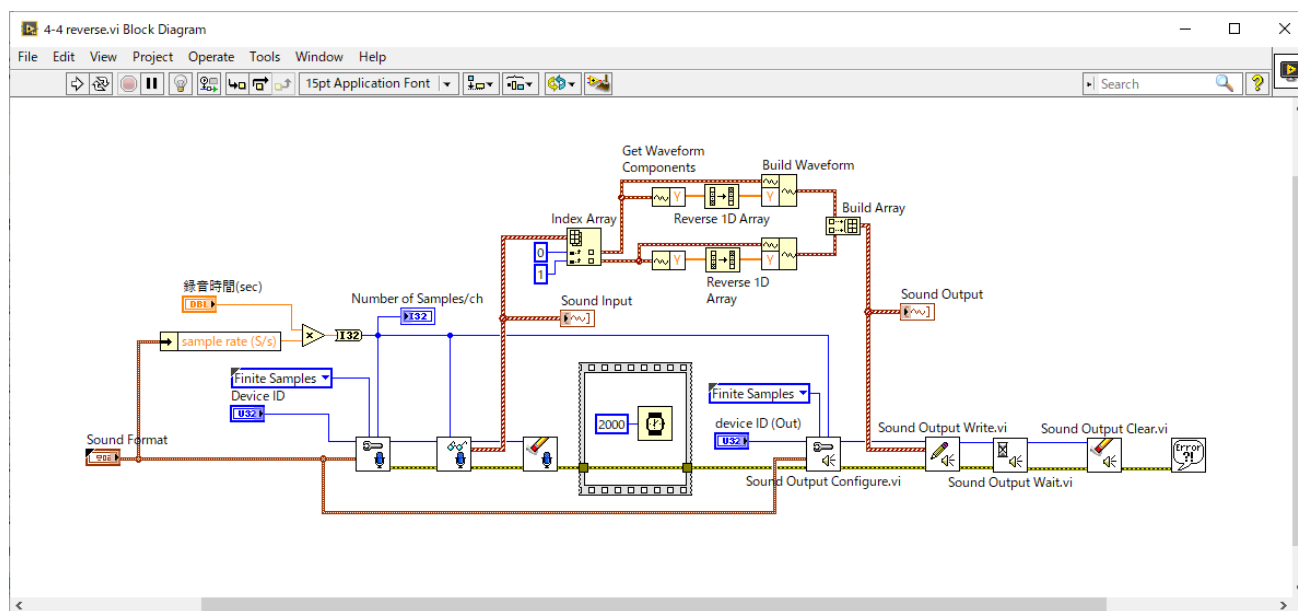


図 4-27 reverse.viのブロックダイアグラム

## 4.5 Forループとシフトレジスタと配列

この辺で少し録音再生のアプリケーション作りから離れて、**Forループ**と配列について説明します。Forループと配列が自由自在に操れるようになるとLabVIEWでのプログラミングが楽になります。

図4-28はForループを使った1からNまでの合計を求めるブロックダイアグラムです。Forループは紙が何枚か重なったような絵で表され、この中に入れたコードが指定した回数、繰り返し実行されます。繰り返し実行する回数をカウント端子に入力します。**シフトレジスタ**はループの実行結果を次のループに伝えるための仕組みで、右端のシフトレジスタ端子に入力された値が、次のループで左端のシフトレジスタ端子から出力されます。ここではループ外部から左端のシフトレジスタ端子に初期値として「0」を入力しているので、1回目のループ動作時に左端のシフトレジスタ端子から「0」が出力されます。初期値を接続しない特殊な使い方もありますが、基本的には初期化して使います。

「i」と書かれた**反復端子**はループ実行毎に0からカウントアップします。シフトレジスタ出力と反復端子の値に「1」を加えたものを加算します。その結果をシフトレジスタの入力端子に接続して次のループで使用します。N回繰り返し実行すると1からNまでの合計が得られる仕組みです。計算の経過を確認できるようにトンネル（指標付け）を使用しました。トンネル（指標付け）ではループの実行毎の値を配列に出力します。ループが10回繰り返し実行した時には要素数10の配列となります。ループが10回繰り返し実行した最後の値だけが欲しい場合もありますので、その場合は「最終値」を

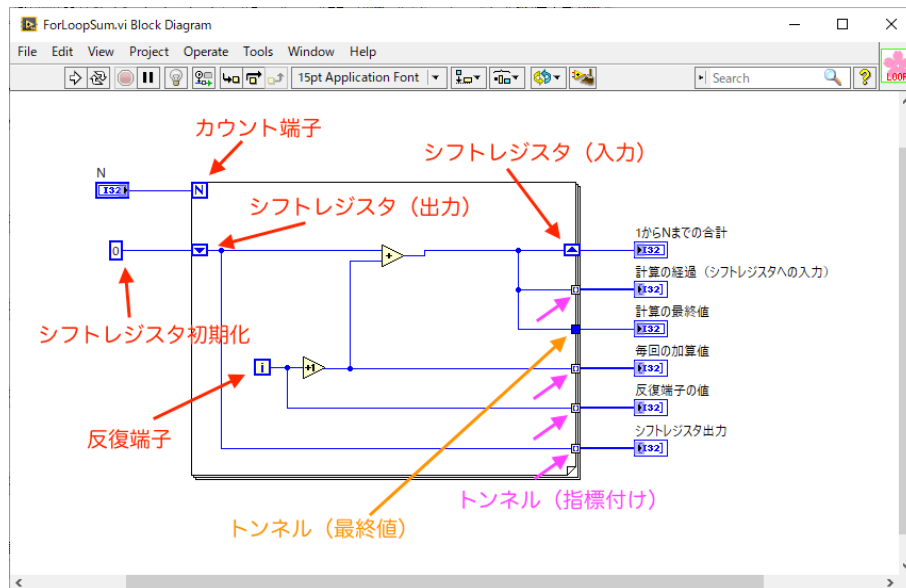


図 4-28 1からNまでの合計を求める



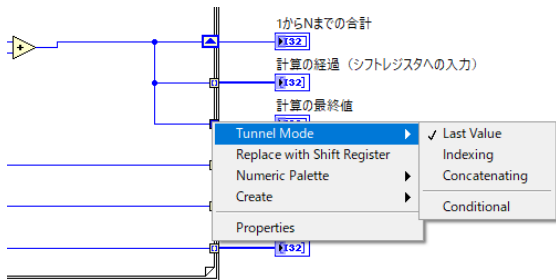


図 4-29 トンネルモードの選択

選択することができます。トンネルが塗りつぶされているのが「最終値」、白い四角の中にカギカッコが入っているのが「指標付け」です。Forループでは「指標付け」がデフォルト設定ですが、トンネルを右クリックして「Last Value (最終値)」を選択することができます (図4-29)。N=10で実行すると図4-30のようになります。ハイライト実行で

**ForLoopSum.vi**のプログラムの動作を確認してください。

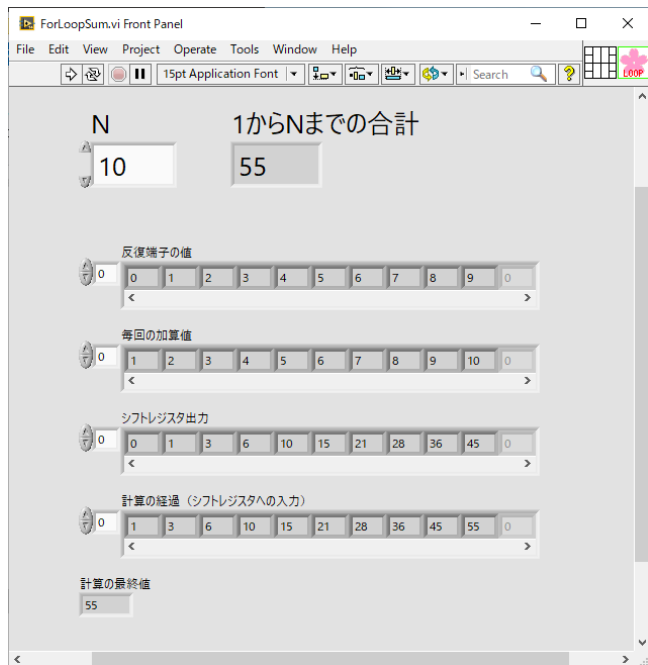


図 4-30 N=10での実行結果

トンネルモードの「条件付き (Conditional)」オプションについて、N以下の素数をリストアップする**エラストテレスの篩**を例に使い方を考えてみます。素数は、1と自分自身でしか割り切れない2以上の自然数です。

エラストテレスの篩は、「Nの平方根以下の素数が既知である時に、N以下の数から既知の素数の倍数を全て取り除けば良い」というものです。N=10000で具体的に考えると、101から10000の間にある素数は、100以下の素数の全ての倍数を取り除いたものである。Nが10000でも、平方根である100まで処理すればいいということですから、強力な指針で感動を覚える人が多いと思います。

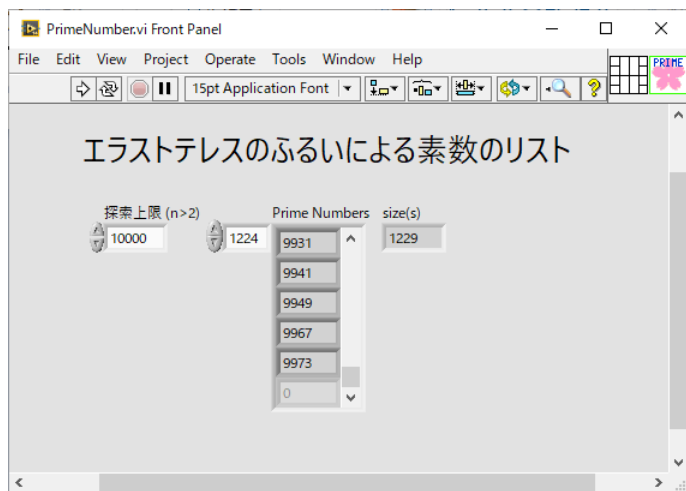


図 4-31 素数をリストアップするエラストテレスの篩

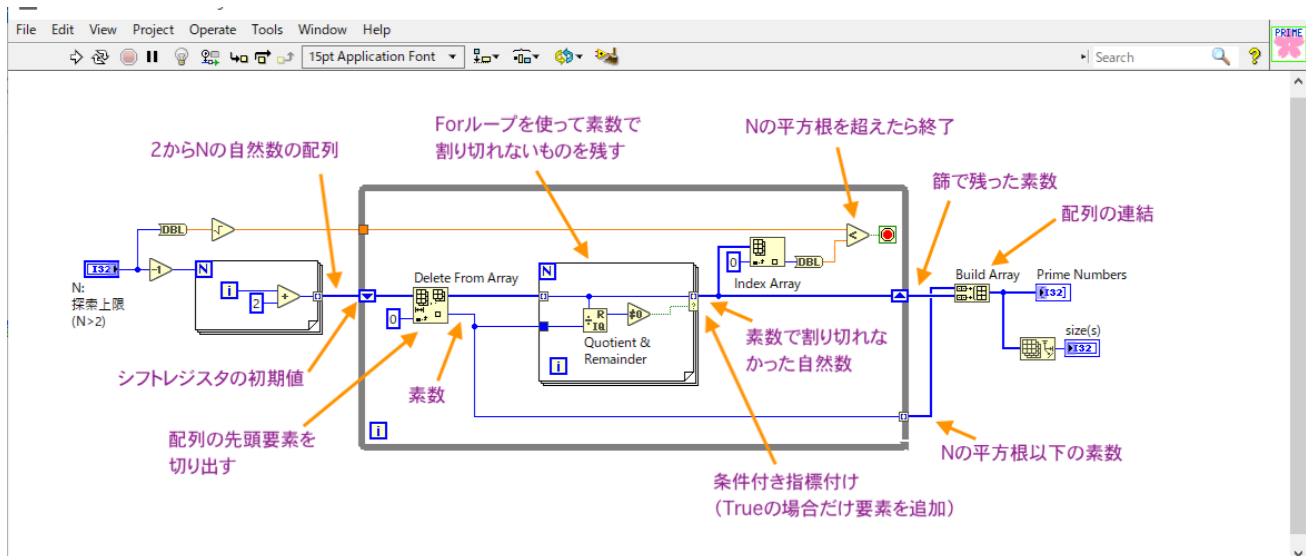


図 4-32 PrimeNumber.viのブロックダイアグラム

PrimeNumber.viのフロントパネルは図4-31で、どこまでの素数をリストアップするか指定して実行ボタンを押すと結果が配列で出力されます。ブロックダイアグラムは図4-32です。左のForループで2から上限値までの自然数の配列ができます。この配列を中央のWhileループのシフトレジスタに初期値として入力します。Whileループの中で**Delete From Array**関数を使ってシフトレジスタの配列から指標0の要素を削除します。削除した指標0の要素は**Delete From Array**関数の端子から取り出すことができます。Whileループの1回目の実行時（反復端子「i」が0）の指標0の要素は2で最初の素数です。Forループには3からNまでの自然数が要素の配列が入力され、素数2で割り切れないものだけが配列として出力されます。2の倍数が篩い落とされた結果、この配列の指標0は3です。Whileループの2回目の実行時（反復端子「i」が1）にシフトレジスタから出力される配列の指標0の要素は3で素数です。今度はForループで3の倍数が篩い落とされて、、、という具合に素数の倍数がふるい落とされる仕組みです。Forループを詳しく見ると、「条件付き指標付け」を使って、素数で割り切れないものだけを配列で出力します。このForループのカウント端子にはループ回数を指示する数値が接続されていませんが、配列が指標付きトンネルで接続されていますので、要素数だけ繰り返し実行されます。先頭は素数となりますので、Nの平方根を超えているかどうかチェックします。超えていたら終了、超えていなければ先頭の素数で処理を繰り返します。

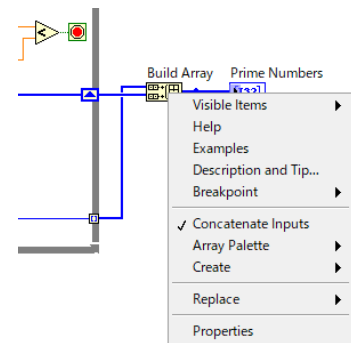


図 4-33 配列の連結

WhileループからNの平方根以下の素数と、節で残った素数が出てきますから、Build Arrayの設定を**連結 (Concatenate Inputs)** (図4-33) にして1次元配列として連結します。

配列の左上にある指標表示を下に引き伸ばすことで配列の次元を2次元、3次元、4次元と増やすことができます。面の情報を表現する時に2次元配列がよく使われます。円錐の関数を表示した例を図4-34に示します。Forループを2重にして行と列がそれぞれ要素数300の2次元配列に関数の値を書き込みました。通常のように2次元配列を数値で示したものの他に、**強度グラフ (Intensity Graph)** や**3D Graph**、**3D Mesh**など用途によって使い分けことができます (図4-35)。3D Graph、3D Meshは制御器パレットでグラフ (Graph) パレットの3D Graphから取り出すことができます。それぞれのグラフには多数のパラメータがありますのでヘルプを見て研究してください。

## 4.6 波形生成とサブVI

録音再生のアプリケーションに戻る前に、波形生成関数を使ってスピーカーから音を発生させるプログラムを例にとってブロックダイアグラムの一部をサブVIにまとめる方法について説明します。ピポパで馴染まれている電話のトーンダイヤルの音は高い音4種、低い音4種の組み合わせで16個のキーを識別する方式で**DTMF (Dual-Tone Multi-Frequency)** と呼ばれる国際標準になっています。**PiPoPa.vi** (図4-36) は電話機のプッシュキーを文字列で入力するとスピーカーからトーン信号を発生するプロ

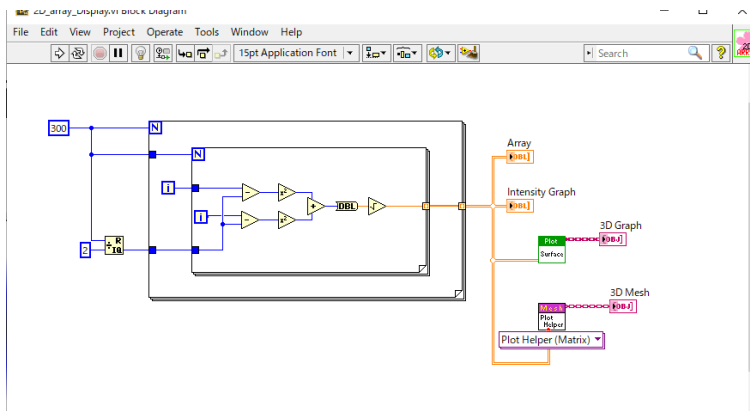


図 4-34 2次元配列の表示

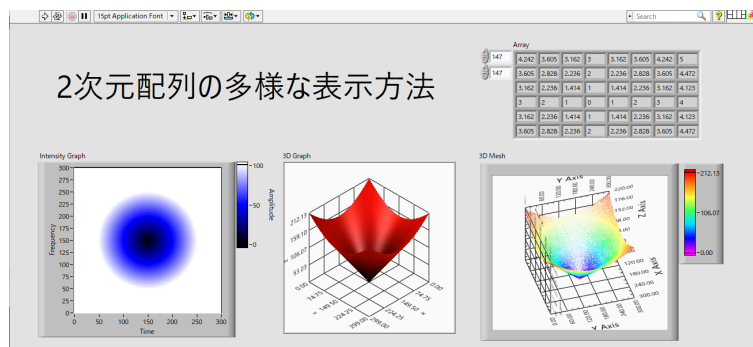


図 4-35 各種グラフ

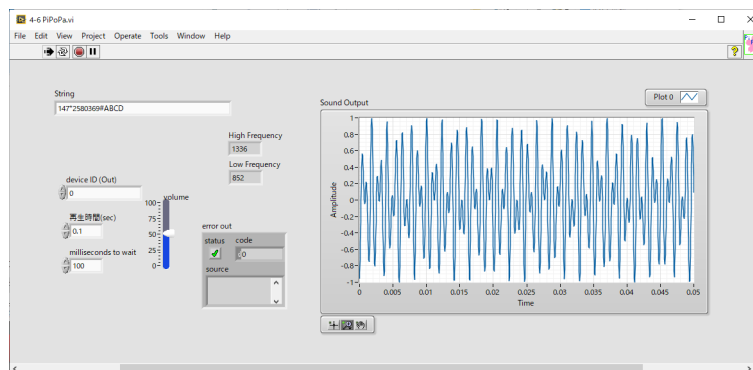


図 4-36 トーン信号を発生するプログラム

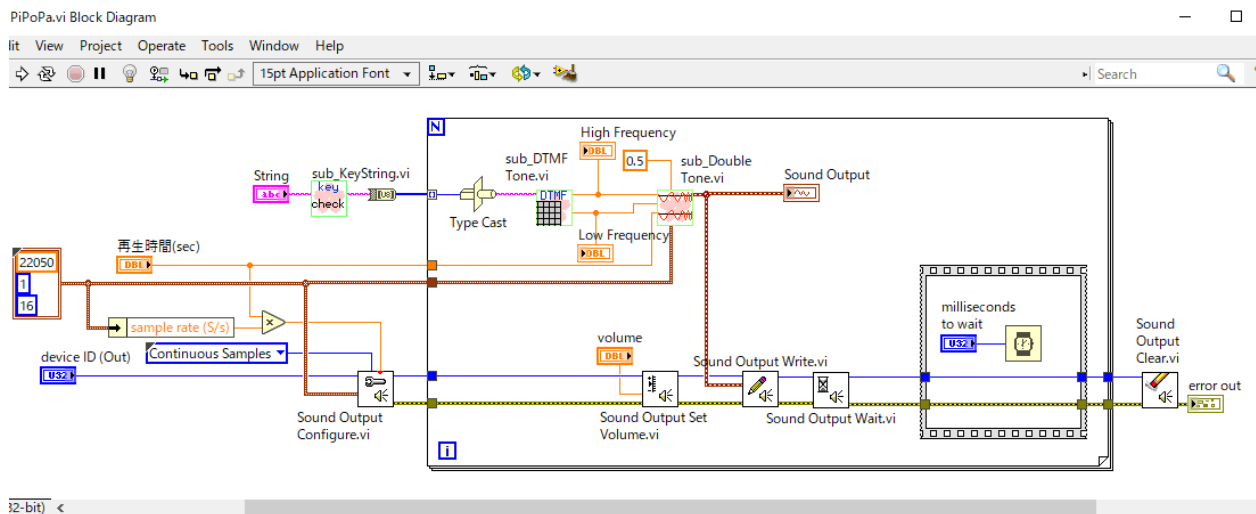


図 4-37 ブロックダイアグラム

グラムです。「String」に0-9、A-D、\*、#の16種類の文字を入れて実行ボタンを押すと音が出ます。いきなり大きな音が出てはびっくりしますのでVolumeの値を低めに設定してから試してください。ブロックダイアグラムは図4-37です。この中で**Type Cast (型変換)** 関数を使ってU8を文字列に変換しています。コンピュータの中でデータ型がどのように扱われているか考えながら、ヘルプを読んで研究して使ってください。Forループの中で一音ずつ発生します。

**sub\_KeyString.vi**は「String」から入力された文字列からプッシュキー以外のものを除去するサブVIです（図4-38）。文字を0-255 (U8) のASCIIコードに変換してプッシュキーに該当するものだけ条件付き指標付けで出力します。

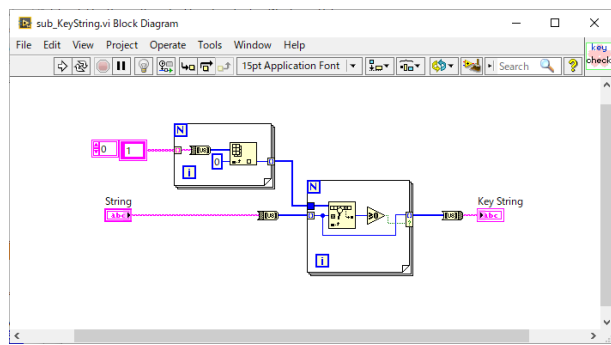


図 4-38 入力文字列のフィルター

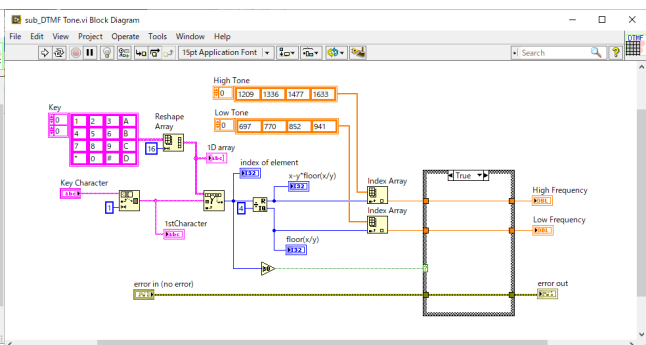


図 4-39 キーに対応する周波数の出力

図4-39はプッシュキー文字に対して一組の高音・低音の周波数を出力します。「Key」がプッシュフォンのキー配置になっていますが、縦の列に「High Tone」、横の列に「Low Tone」が割り当てられます。例えば「6」のキーであれば「High Frequency」が1477Hzと「Low Frequency」が770Hzがで出力されます。図4-40がsub\_DoubleTone.viのブロックダイアグラムで、高音・低音のサイン波を作ります。波形生成の関数は、図4-41のシグナルプロセッシング (Signal Processing) パレットの波形生成 (Waveform Generation) パレットに揃っています。その中でSine Waveform.viを使用しました。

サンプル数とサンプルレートのサンプリング情報 (sampling info)、周波数 (frequency)、信号の大きさ (amplitude) を入力しました。波形データの加算で二つの信号を合成します。フロントパネルは図4-42です。サブVIを作る時には入力と出力の端子を設定します。端子を設定する前は図4-43のようにデフォルトアイコンの左側に空白の枠があります。ブロックダイアグラムに配置して配線した時に違和感がないように、入力は左側の端子、出力は右側の端子、エラーは一番

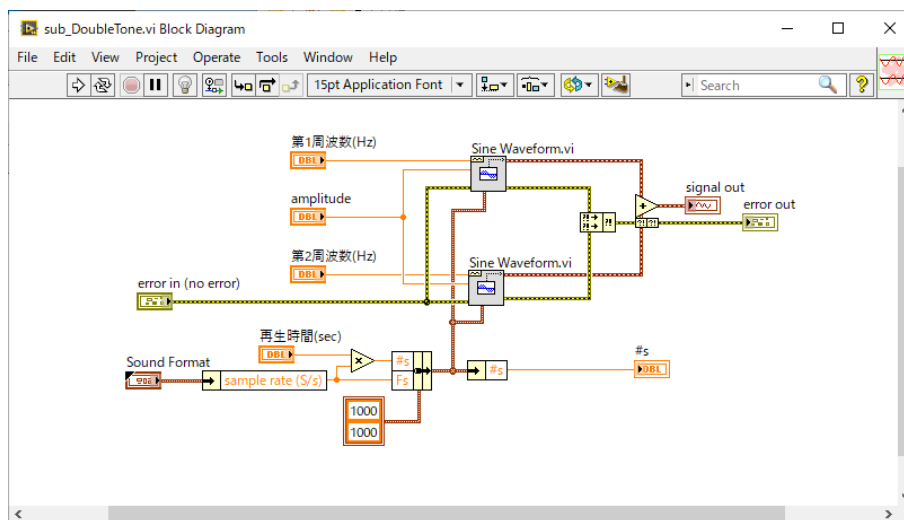


図 4-40 高音・低音のサイン波の生成

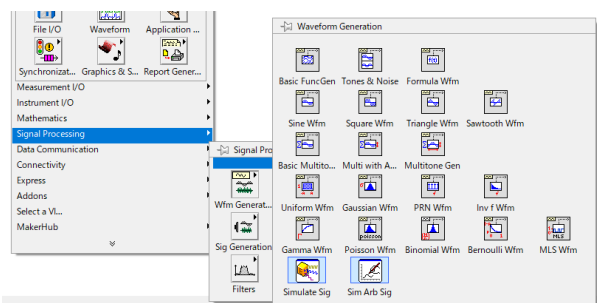


図 4-41 波形生成 (Waveform Generation) パレット

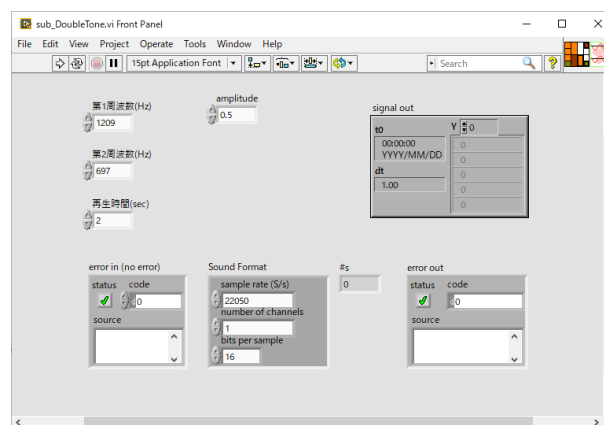


図 4-42 sub\_DoubleTone.viのフロントパネル

下の端子というように暗黙の了解があります。図4-44は左上の端子をクリックした後で「第1周波数(Hz)」をクリックした状態です。このようにして端子と制御器・表示器を関連付けます。関連付けた端子を取り消す時には端子を右クリックして「この端子を切断 (Disconnect This Terminal)」を選択します (図4-45)。アイコンもオリジナルなデザインを作ることができます。このテキストで使っているVIは統一感を出すために桜の花をベースにしたアイコンにしましたが、機能が一目でわかるというところまでは至っていません。アイコンをダブルクリックすれば図4-46のアイコンエディターが開きますのでレイヤー (Layer)、グリフ (Glyphs)、アイコンテキスト (Icon Text)、テンプレート (Templates) を駆使して、機能が一目でわかるアイコンを作ってください。

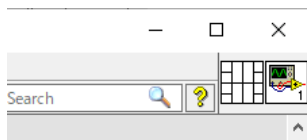


図 4-43 サブVIの入出力端子

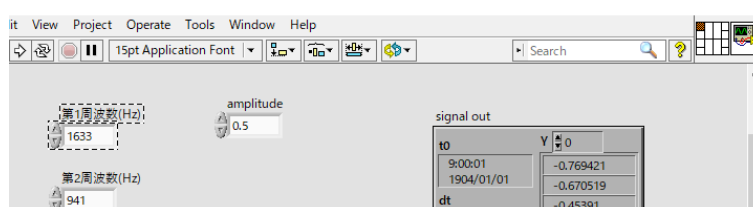


図 4-44 端子と制御器・表示器の関連付け

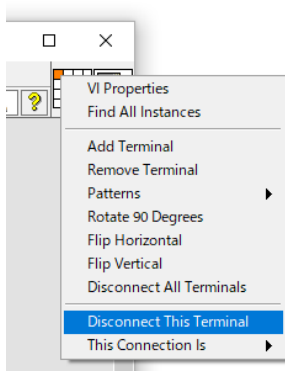


図 4-45 この端子を切断

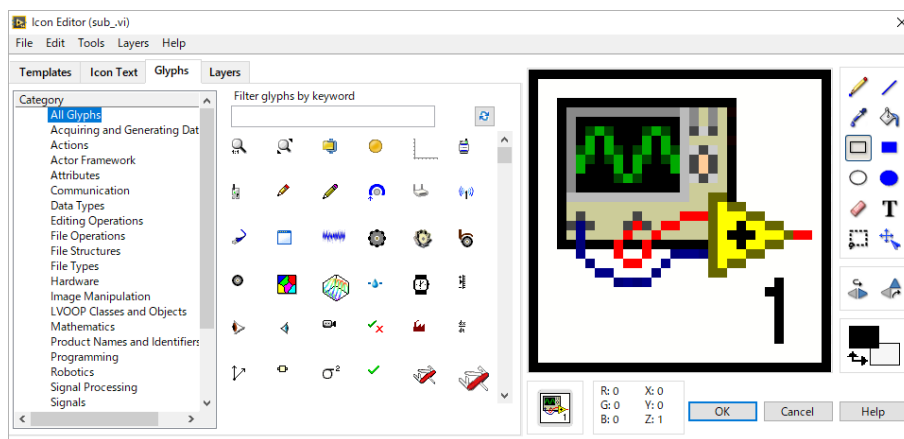


図 4-46 アイコンエディター

## 4.7 録音、再生プログラム

録音・再生、ファイルへの読み書きができるアプリケーションを作りましょう。SoundVIEWのシンプルバージョンという感じです。ステートマシンで、初期化、録音、再生、ファイル保存、ファイル読み込み、終了の6個の状態を考えます。

ファイルメニューから新規VI (New VI) をクリックします。**easyRecorder.vi**という名前で保存します。フロントパネルに列挙体 (Enum) を作成します (図4-47)。名前を**my\_state**と付けておきます。

列挙体 **my\_state** を右クリックし「項目を編集 (Edit Items...)」を選択します (図4-48)。図4-49のように6個の項目名を記入します。列挙体 **my\_state** を右クリックし Make Type Def. (タイプ定義を作成) を選択します (図4-50)。列挙体 **my\_state** を右クリックし Open Type Def. (タイプ定義を開く) を選択します (図4-51)。

**my\_state.ctl**という名前で保存して、閉じるとフロントパネ

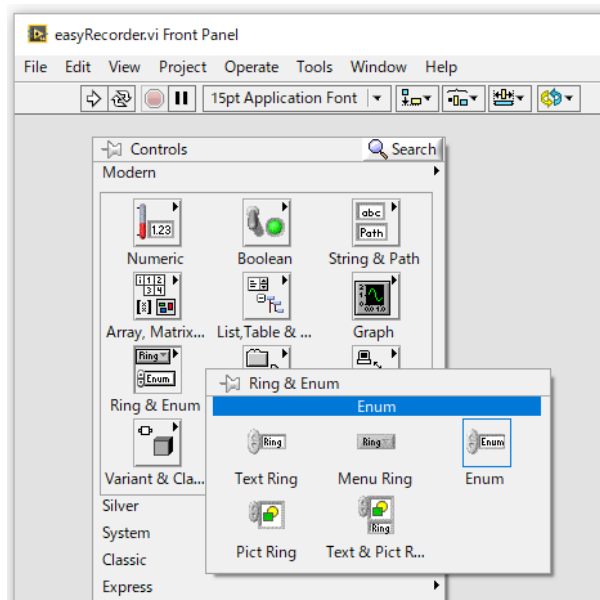


図 4-47 列挙体 (Enum) を作成

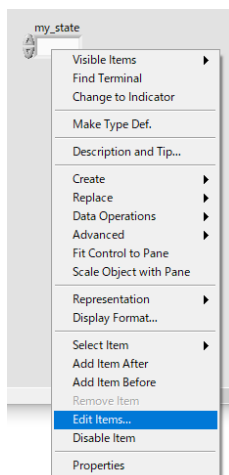


図 4-48 項目を編集

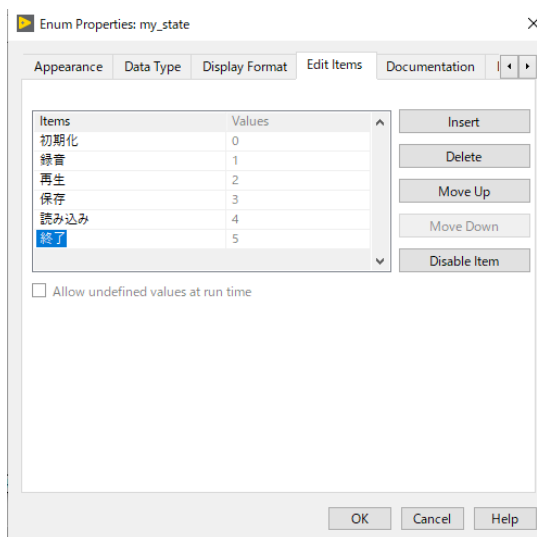


図 4-49 項目名の記入

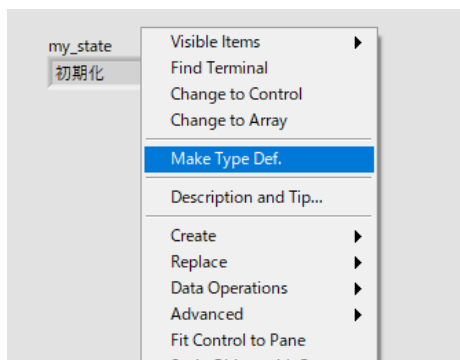


図 4-50 Make Type Def.

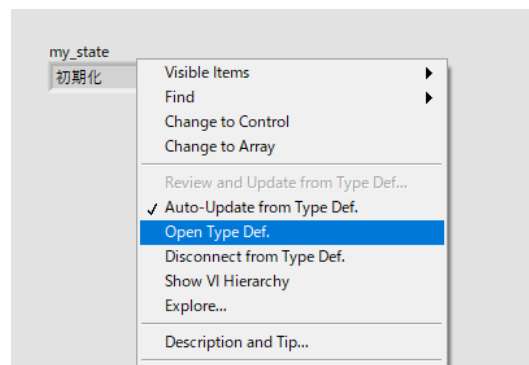


図 4-51 Open Type Def.

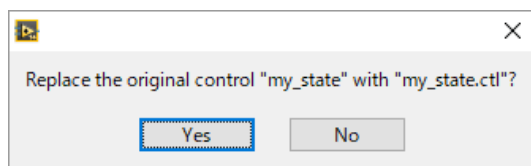


図 4-52 確認のダイアログ

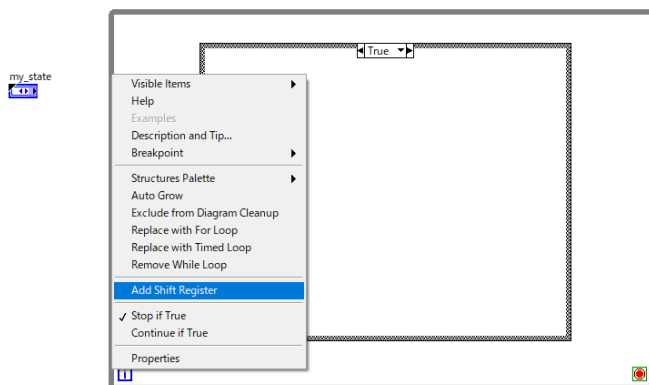


図 4-53 シフトレジスタを作成

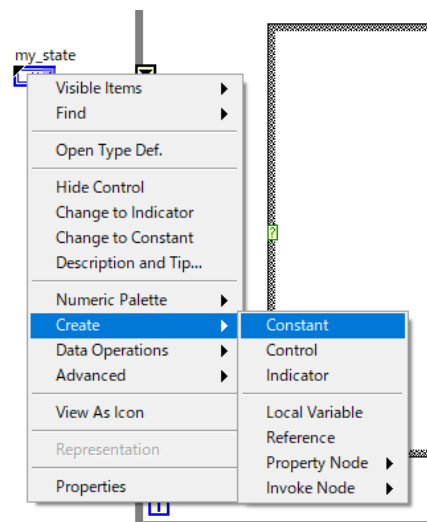


図 4-54 列挙体の定数を作成



ルの列挙体 **my\_state** を置き換えるかという確認のダイアログが出るので、「Yes」を押します（図4-52）。Whileループにシフトレジスタを作ります（図4-53）。図4-54のようにタイプ定義（Type Def.）した列挙体 **my\_state** を右クリックして定数（Constant）を作成します。Whileループの外側で **my\_state.ctl** の定数の「初期化」を接続します。ケースストラクチャのセレクト端子とシフトレジスタを結線するとサブダイアグラムのケースセクタラベルに **my\_state.ctl** の「初期化」と「録音」が表示されます。図4-55のように「録音」サブダイアグラムを表示してケースストラクチャの外枠上で右クリックしポップアップメニューから **Add Case After** を選択し、ケースを追加します。同様にケースを追加して全てのケースを作成します。6個のサブダイアグラムができました（図4-56）。 **Add Case After**

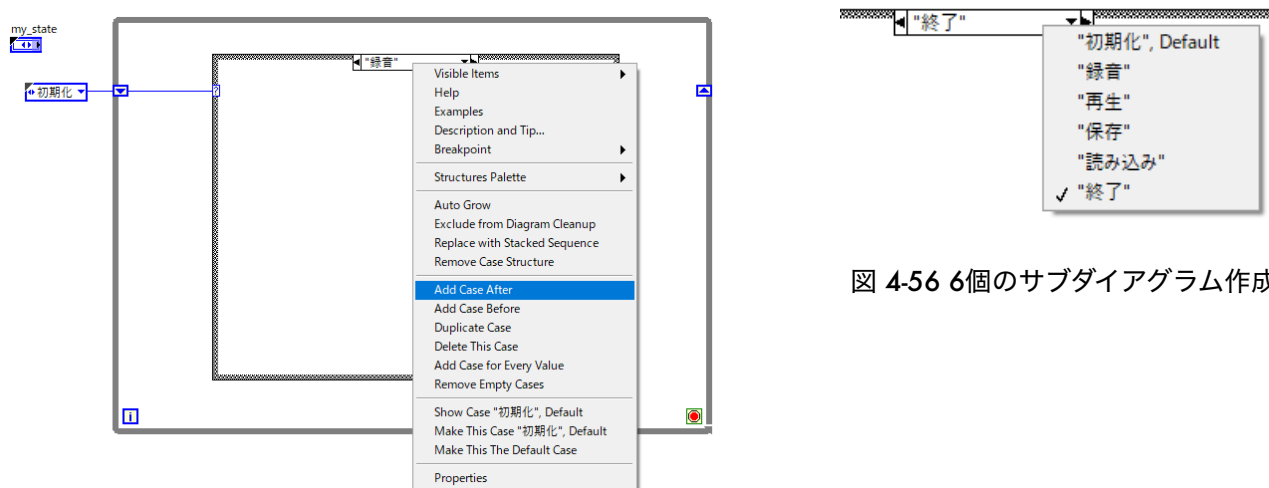


図 4-56 6個のサブダイアグラム作成

図 4-55 Add Cae Afterを選択し、ケースを追加

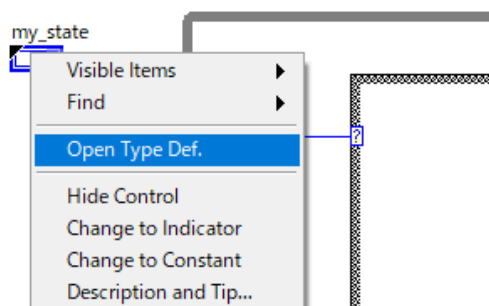


図 4-57 タイプ定義を開く

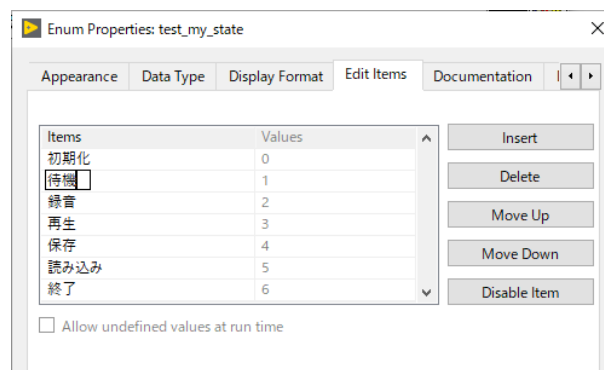


図 4-58 「待機」を追加

の4項目下の**Add Case for Every Value**を選択すると一度に全てのサブダイアグラムが作成されます。

うっかりしていましたが、「待機」のステートも必要です。

図4-57のように、タイプ定義した列挙体 **my\_state** を右クリックして「Open Type Def.」を選択します。列挙体 **my\_state** のタイプ定義が開きますので「Insert (挿入)」ボタンで「初期化」項目の後に「待機」項目を追加します (図4-58)。図4-59のように、シフトレジスタに接続していた列挙体 **my\_state** の定数も更新されています。ステートマシンでは列挙体の定数が多く使われますが、タイプ定義を行うことで将来の変更に容易に対応できることを実感してください。

サブダイアグラム「初期化」項目の後に「待機」項目を追加します (図4-60)。

フロントパネルにボタンやグラフを追加します (図4-61)。「待機」サブダイアグラムではボタン操作に応じたサブダイアグラムへ移動できるようにシフトレジスタに出力します (図4-62)。

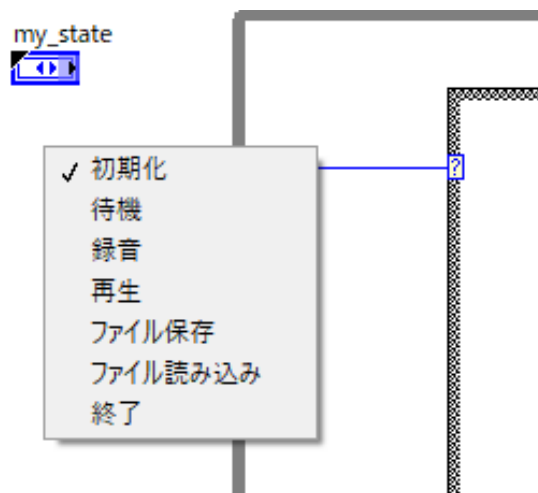


図 4-59 列挙体 **my\_state** の定数も自動的に更新

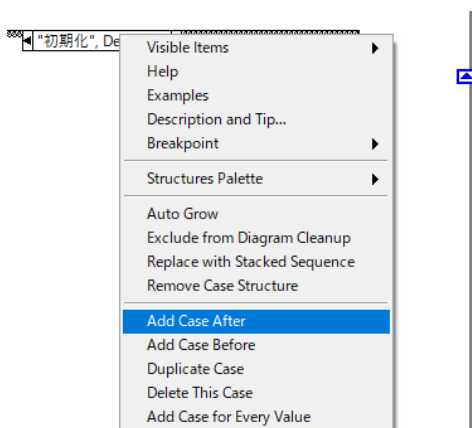


図 4-60 「初期化」項目の後に「待機」項目を追加

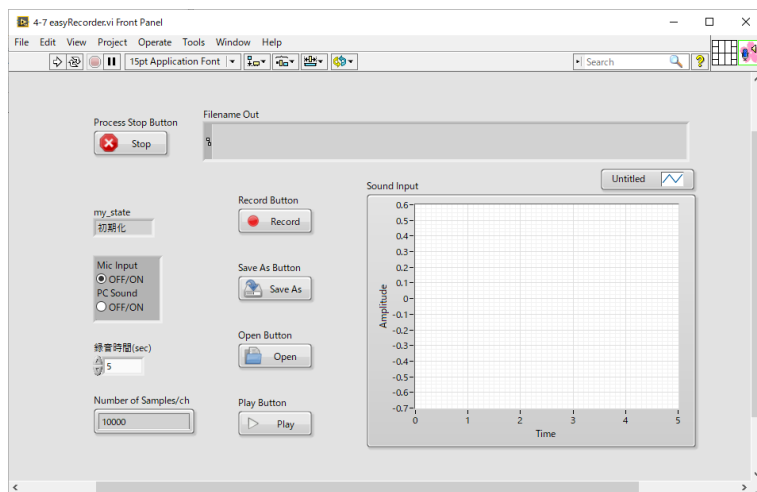


図 4-61 フロントパネルに制御器・表示器を追加

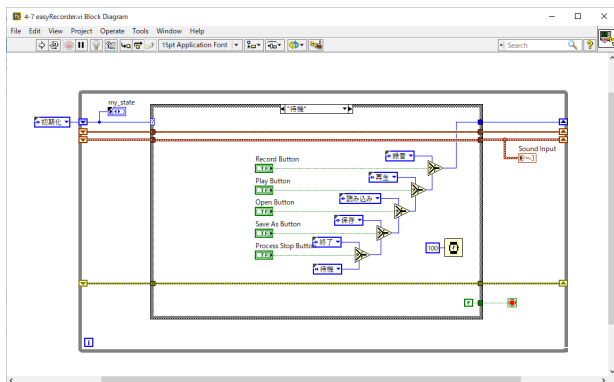


図 4-62 「待機」サブダイアグラム

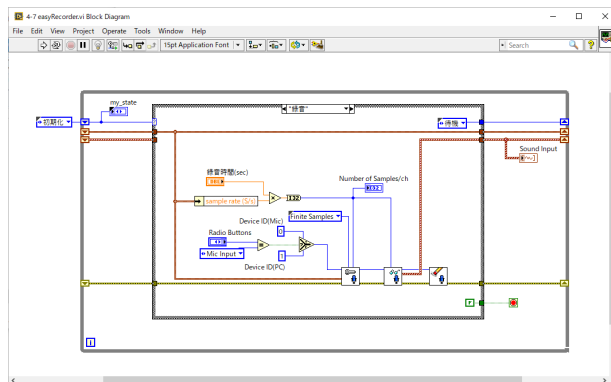


図 4-63 「録音」サブダイアグラム

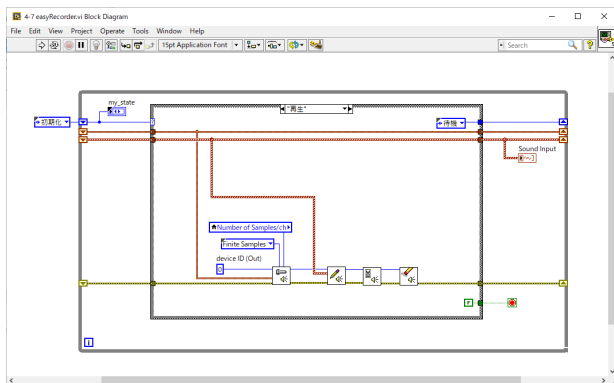


図 4-64 「再生」サブダイアグラム

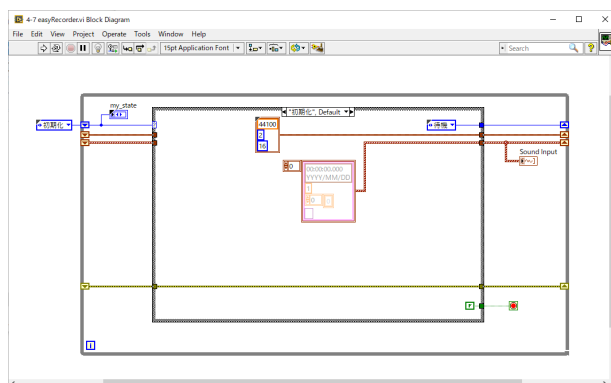


図 4-65 「初期化」サブダイアグラム

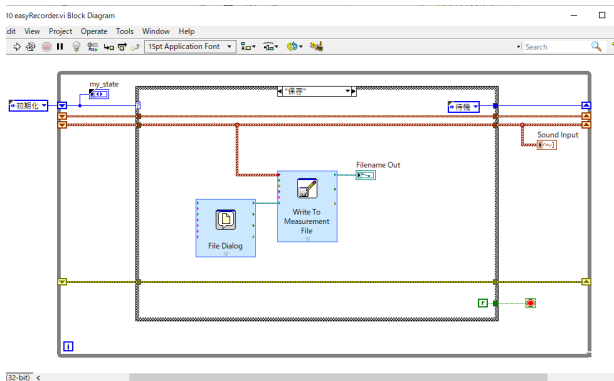


図 4-66 「保存」サブダイアグラム

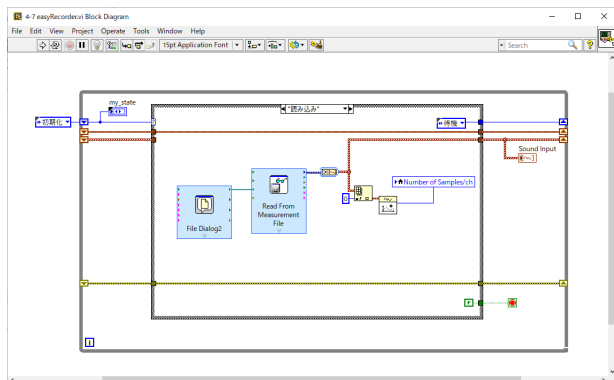


図 4-67 「読み込み」サブダイアグラム

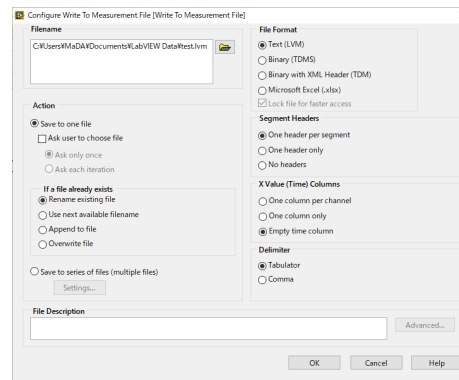


図 4-68 Write to Measurement  
Fileの設定画面

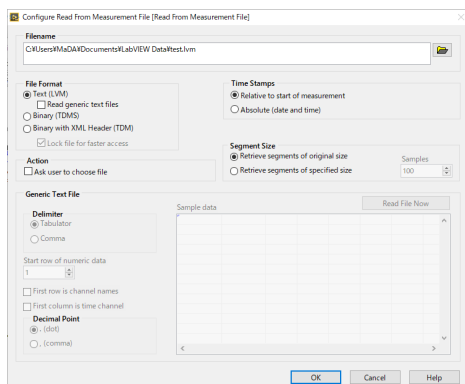


図 4-69 Read from  
Measurement Fileの設定画面

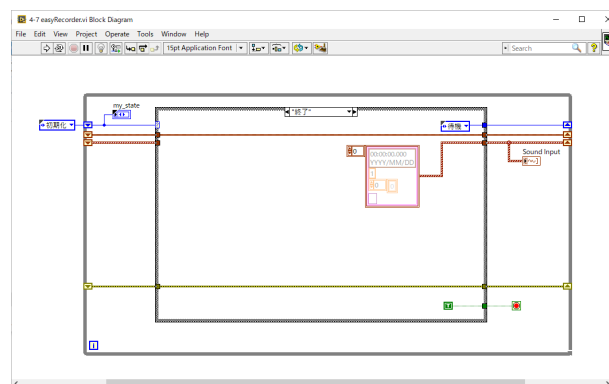


図 4-70 「終了」サブダイアグラム

「録音」サブダイアグラム（図4-63）と「再生」サブダイアグラム（図4-64）はparrot.viの録音部分と再生部分をコピーして作ります。設定パラメータなどは「初期化」サブダイアグラムにまとめます（図4-65）。

「保存」サブダイアグラム（図4-66）と「読み込み」サブダイアグラム（図4-67）では「File I/O」パレットの**Write to Measurement File**と**Read from Measurement File**を使用しました。**Write to Measurement File**と**Read from Measurement File**のような水色の大きなアイコンは**Express VI**と呼ばれています。アイコンをダブルクリックすると設定画面が表示されてパラメータの設定をすることができます。図4-68が**Write to Measurement File**の設定画面で、図4-69が**Read from Measurement File**の設定画面です。テキスト形式での保存はスプレッドシートなどで開くことができるので便利ですが、バイナリ形式よりもファイル容量が大きくなりがちです。研究して用途にあった設定を見つけてください。図4-70が「終了」サブダイアグラムです。実行中のフロントパネルは図4-71です。

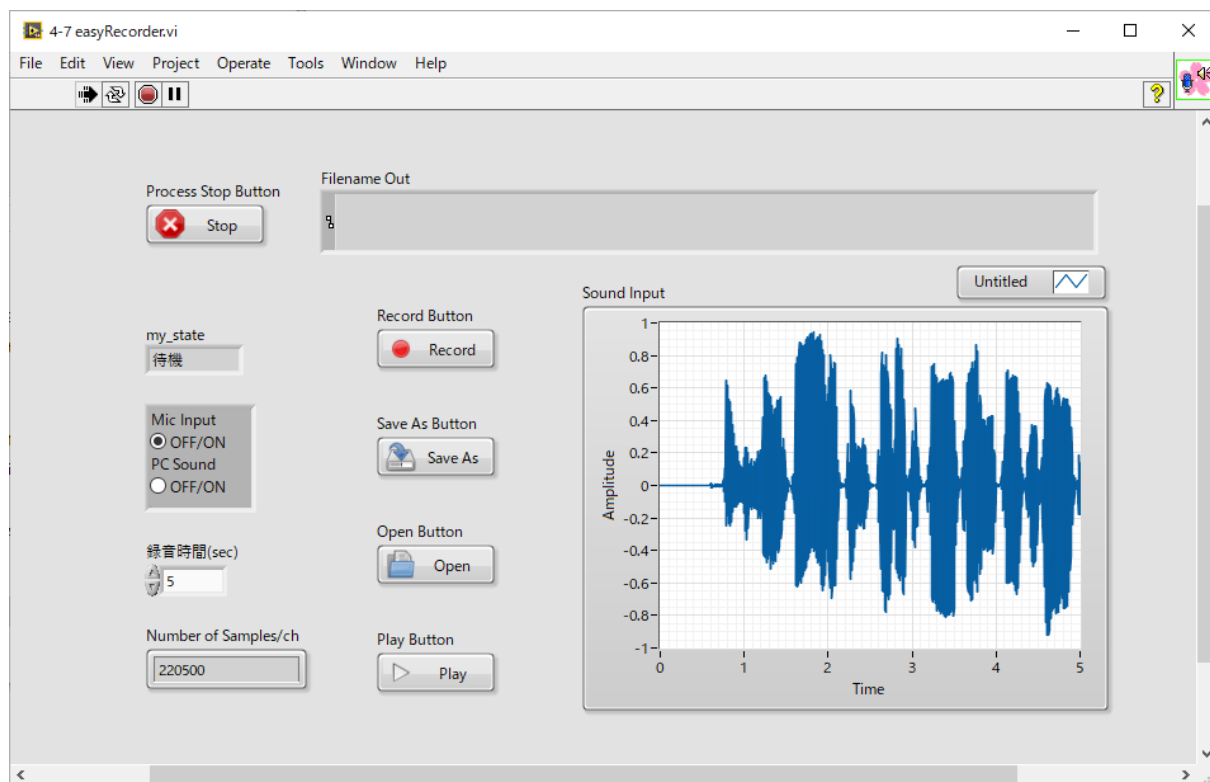
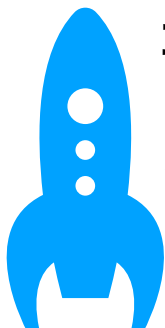


図 4-71 実行中のフロントパネル



## コラム 4 NXGのWEB親和性

皆さんもよく見るホームページですが、実はホームページを作るためには様々なプログラミング言語を使わなくてはなりません。HTML、CSS、Javascriptなどなど…これらを組み合わせて一からホームページを作ろうとするととても大変です。LabVIEW NXG Community Editionには、「Web Module」(図C4-1)というアドオンソフトが付属していて、簡単にホームページを作ることができます。ホームページはHTMLというプログラミング言語がベースになりますが、Web Moduleを使ってVIを作ると、自動的にHTMLコードが生成されます。あとは枝葉の部分を少し手直しするだけです。VIを作る感覚でホームページを作ることができます。

さらにホームページを世界に公開するためには、24時間365日稼働しているパソコンで、作成したホームページアプリを実行し続けなくてはなりません。これを自宅で行うとすると電気が非常にかかります。Web Moduleには「Systemlink Cloud」(図C4-2)というソフトウェアのライセンスも付属しています。今はやりのクラウドですね。ホームページアプリをクラウドにアップロードすると、クラウド上でアプリが実行されるので、アクセスするための手順さえ分かれば、外部からホームページアプリへアクセスできます。もちろんセキュリティ設定もありますので、知っている人しかアクセスできないようにもできます。

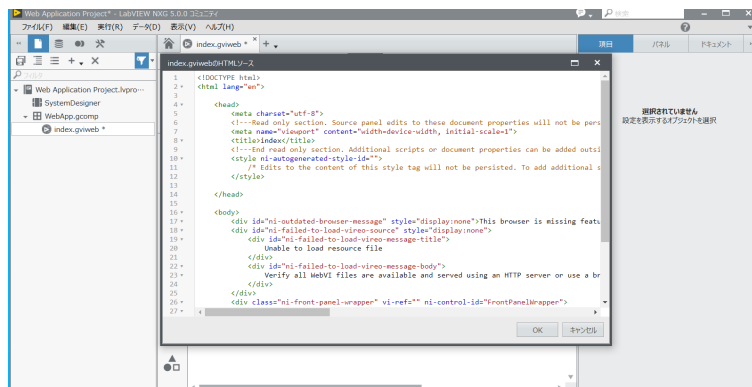


図 C4-1 LabVIEW NXG Web Module

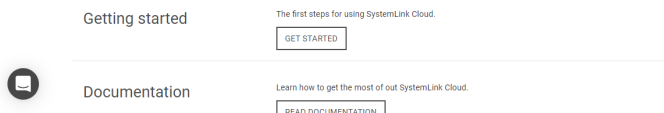
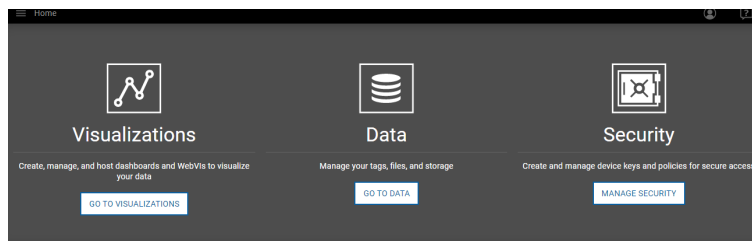


図 C4-2 Systemlink Cloud

## 第 3 部

# LabVIEWとArduinoで 電子工作

電子工作が初めての方には「みんなのArduino入門：基本キット」  
(スイッチサイエンス) の購入をお勧めします。

---

## 第 5 章

# LabVIEWと Arduino



Arduinoを使ってフィジカルコンピューティングを始めましょう。ArduinoにLINXファームウェアを書き込んでLabVIEWでプログラムを作ります。デジタル入出力とアナログ出力(PWM)を使って押しボタンスイッチでファンの風量を制御するプログラムを作ります。

**[キーワード]** Lチカ、Arduino IDE、COMポート、ブレッドボード、LINX、LINX Firmware Wizard

**[使用する部品]** Arduino UNO 1個、ブレッドボード 1個、タクトスイッチ 1個、抵抗 (100Ω) 1個、ファン (5V DC駆動) 1個、ワイヤ 4本

## 5.1 Arduino IDEのインストールとLチカ

Arduinoは2005年にイタリアで生まれたマイコンボードです。**オープンソースハードウェア**といって回路図や基板が公開されていて誰でも**互換機**をつくることができたので、瞬く間に世界中で使われるようになりました。安価なものは500円ぐらいで購入できます。ここでは、Arduino UNOという一番ポピュラーなボードを使っていきます。プログラムはArduino IDEという無料の開発ツールで作ります。センサやモーターなどを使った製作例を紹介する記事が豊富なので、使いながら使い方を覚えていくことができます。使ってみたいセンサなどが出てきたら、Webで「Arduino」+「センサ名」で検索すると、使い方の記事がすぐに見つかるでしょう。

それでは、**Arduino IDE**をダウンロードします。「Arduino CC」で検索すると「<https://www.arduino.cc>」（図5-1）を見つけることができます。「SOFTWARE > DOWNLOADS」でArduino IDEのダウンロード画面（図5-2）に移動して、Windows installerを選ぶと「寄付してダウンロード（CONTRIBUTE & DOWNLOAD）」か「ダウンロードだけ（JUST DOWNLOAD）」かを選択する画面（図5-3）になります。どちらでも意志のままに進んでください。ダウンロードは特に面倒なことはないはずです

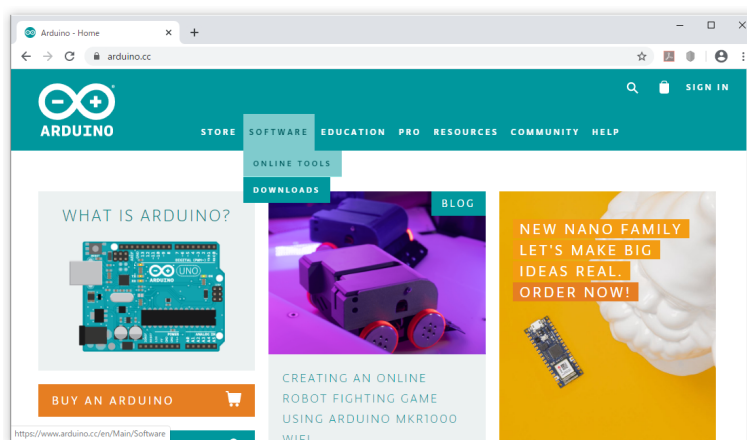


図 5-1 Arduino CCのホームページ



図 5-2 Arduino IDEのダウンロードページ

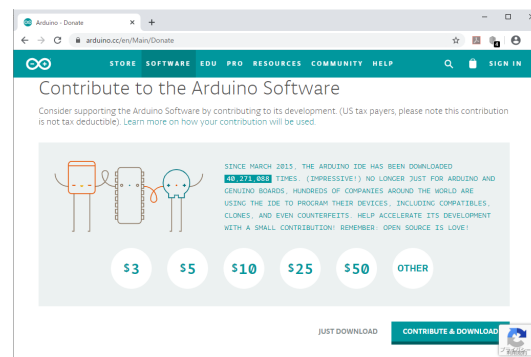


図 5-3 ダウンロード



が、困った時は「Arduino IDE ダウンロード」で検索すると親切なサイトがたくさん見つかります。ダウンロードしたファイルを実行するとインストールが始まります。インストールが済んだらさっそくArduino IDEを起動してみましょう。

Arduinoが使える状態かどうか調べる方法として**チ力**があります。Arduino IDEを開いてファイルメニューから「スケッチ例 > 01.Basics > Blink」を選択します（図5-4）。**Blink.ino**（図5-5）が開きます。

Arduino UNOをUSBポートに接続します。**Blink.ino**ウィンドウのツールメニューのボード設定を「Arduino UNO」にします（図5-6）。ツールメニューのシリアルポートで

「(Arduino/Genuino Uno)」と書かれているポートを選択します（図5-7）。左上にある丸の中に右矢印のあるアイコンをクリックするとプログラムがマイコンボードに書き込まれます（図5-8）。**Blink.ino**はArduino UNOのLED（図5-9のオレンジマーク）を1秒間隔で点滅させるプログラムです。実は、Arduino UNOは工場出荷の時に**Blink.ino**が書き込まれているので、USBポートに接続された時から1秒間隔で点滅していたと思います。正常に書き込まれても、書き込まれなくとも1秒間隔で点滅するので書き込めたかどうかの確認になりません。**Blink.ino**プログラムの、`delay(1000);`と書かれた2行（図5-5の矢印）を`delay(100);`に変更して、右矢印アイコンをクリックしてマイコン

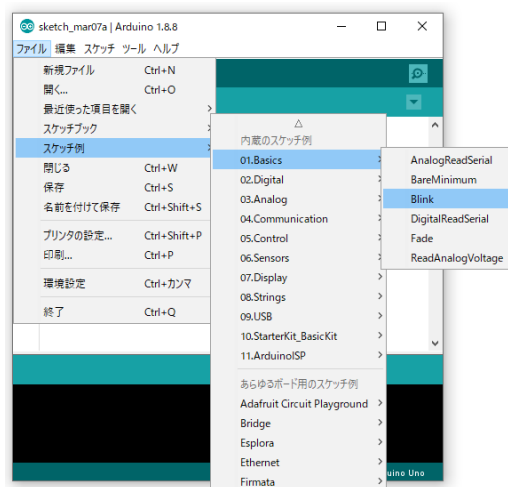


図 5-4 ファイルメニューのスケッチ例

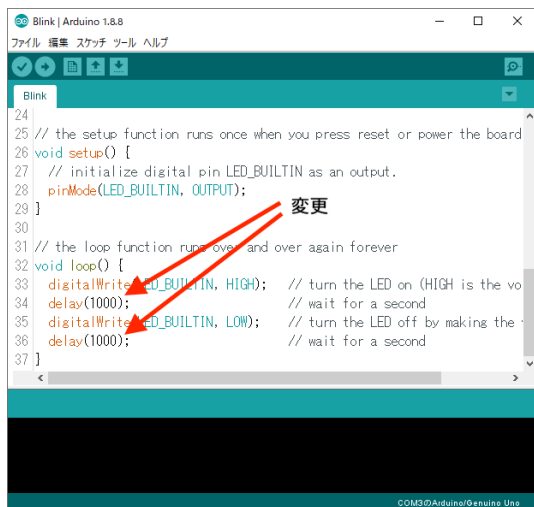


図 5-5 サンプルプログラム Blink.ino

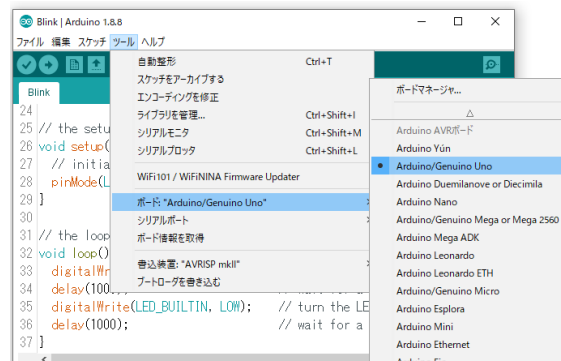


図 5-6 ボードをArduino UNOに設定

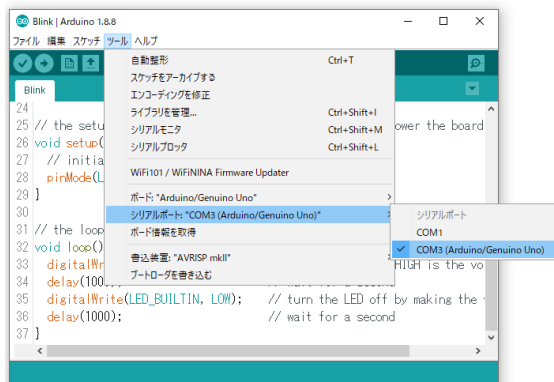


図 5-7 シリアルポートの設定

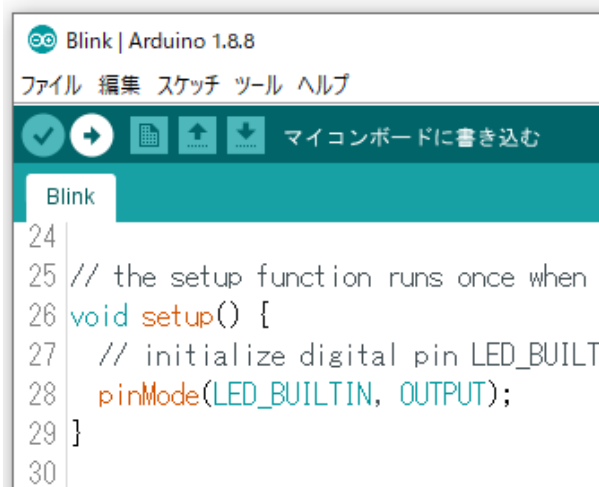


図 5-8 プログラムを書き込む

ボードに書き込みます。点滅が0.1秒間隔に変わったら正常です。LEDがチカチカするので「Lチカ」と呼ばれています。

## 5.2 Arduinoの入力と出力

Arduino UNOの主な入力ピンと出力ピンを図5-9で説明します。図の上側のピンは主にデジタル入出力ピンです。2から13までデジタル入力かデジタル出力に使えます。0ピン、1ピンはUSBでのシリアル通信に使用されるので、一般的にはデジタル入出力ピンとしては使いません。13ピンは基板内部で「Lチカ」で光っていたLEDが接続されています。数字の前にチルダ (~) が書かれているピンはアナログ出力としても使うことができます。アナログ出力は0-255の範囲でLEDの明るさを変えたりファンの回転数を変えることができます。0Vか5Vのデジタル出力ですが、480Hz（5ピンと6ピンは980Hz）のパルスで、パルス幅が256段階で制御できます。図の下側のピンはアナログ入力ピンと電源ピンです。A0からA5が0から5Vのアナログ入力を0-1023までのデジタル値で知ることができます。A4ピンとA5ピンは7章で説明しますが、I2C用のピンとしても使われます。

Arduino IDEを使ったプログラミングの方法をもっと知りたい方は「みんなのArduino入門」（ISBN978-4-89797-948-9）など良書が多数ありますので気に入ったものを選んでください。

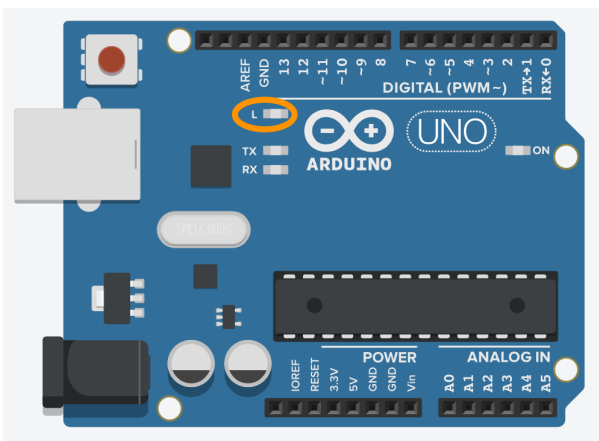


図 5-9 Arduino UNOのLEDとピン配置

## 5.3 LabVIEWでArduinoをコントロール

LabVIEWからArduinoを入出力として使う方法として**LINX**というアドオンツールがあります。LINXはArduinoやRaspberry Pi、BeagleBone BlackなどをLabVIEWのインターフェースとして使うときに便利な仕組みで、Digilent社というNational Instruments社の教育関連事業の子会社が無料で提供しているツールです。**LabVIEW MakerHub**というWebサイト（図5-10）にはLINXの使い方やFAQ、フォーラムなどがありますので、詳しく知りたいときにはアクセスしてみてください。

LINXでは**LINXファームウェア**というプログラムをArduinoに書き込んで、シリアルポートを通してコマンド、データの送受信を行います。LINX関数を使ってLabVIEWでプログラムができる便利な仕組みです。LabVIEW Community Editionでは自動的にLINXがインストールされますが、LabVIEWの他のEditionでLINXを使う場合にはVI Package Managerを起動して「Digilent LINX」をインストールします（図5-11）。便利な反面、コマンドと応答の時間がロスになって高速なデータ収録には向きません。第7章ではLINXを使わないでArduinoと連携する方法について紹介します。

Arduino UNOへのLINXファームウェアの書き込みは**LINX Firmware Wizard**を使います。LINXファームウェアは通常の

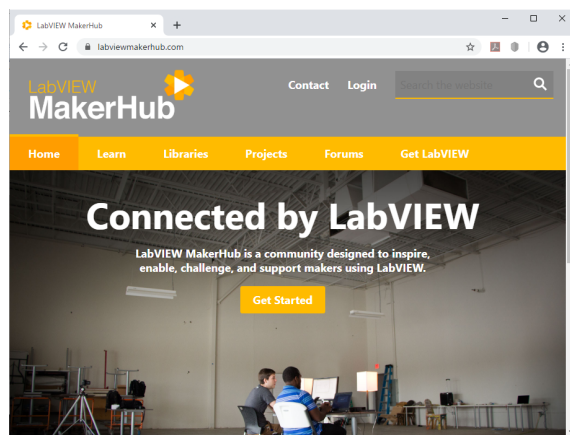


図 5-10 LabVIEW MakerHubのホームページ

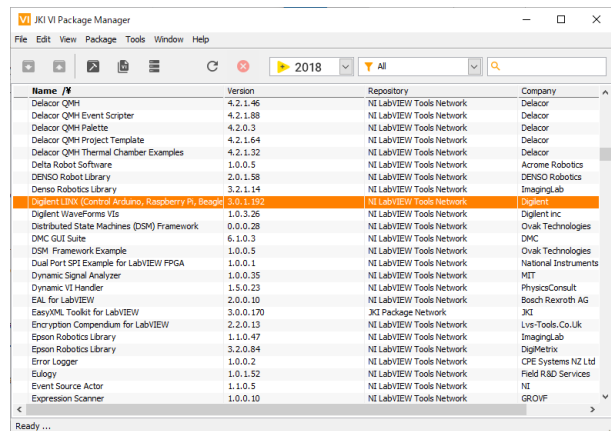


図 5-11 VI Package Manager

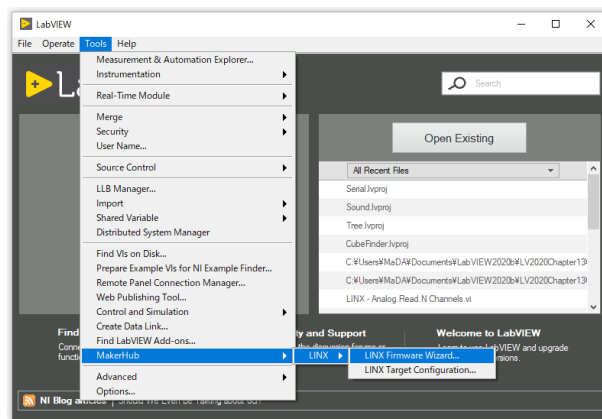


図 5-12 LINX Firmware Wizard..

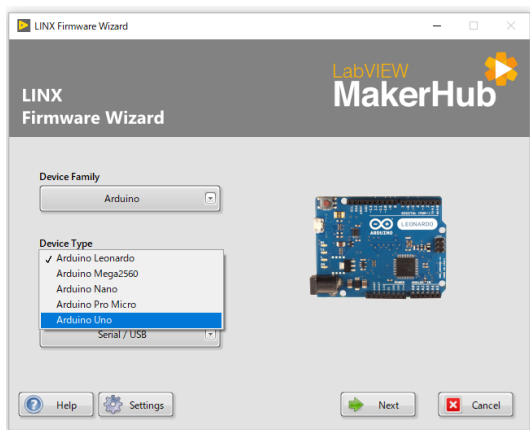


図 5-13 Arduino UNOを選択

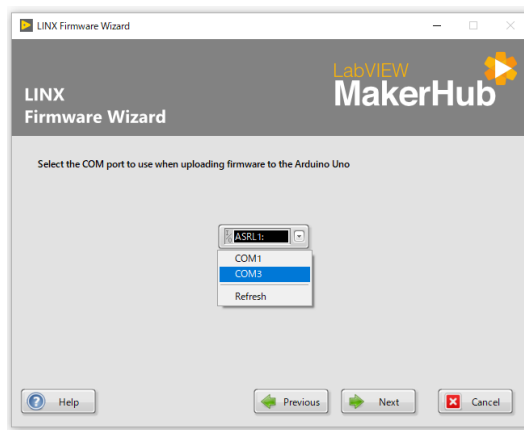


図 5-14 シリアルポートの選択

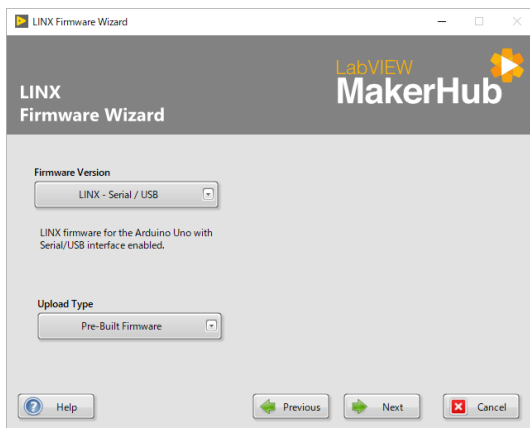


図 5-15 このまま Nextボタン

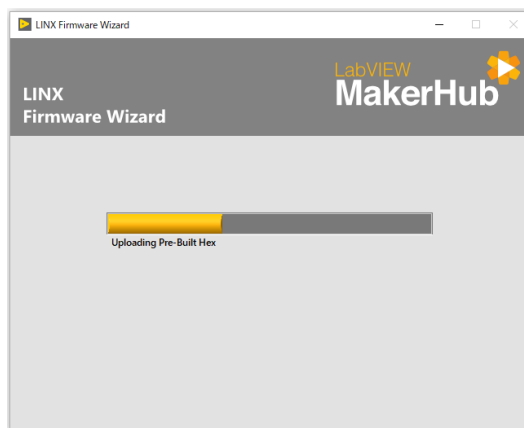


図 5-16 LINXファームウェアの書き込み

Arduinoプログラムですので、**Blink.ino**を書き込んだように他のプログラムを書き込んで他の用途に使うことができます。「Tools」メニューから「MakerHub > LINX > LINX Firmware Wizard...」を選択します(図5-12)。Device Typeで「Arduino UNO」を選んで「Next」ボタンを押します(図5-13)。図5-14のようにUNOが接続しているCOMポートを選択して「Next」ボタンを押します。図5-15のように「Firmware Version」と「Upload Type」の選択画面になりますが、このまま「Next」ボタンを押しますと、図5-16のようにArduino UNOへのLINXファームウェアの書き込みが始まります。書き込みが終了すると図5-17が表示されます。「Launch Example」ボタンを押してください。**LINX - Blink (Simple).vi**が表示されます。シリアルポートを選択して、「Digital Output Channel」は「13」のままで実行ボ

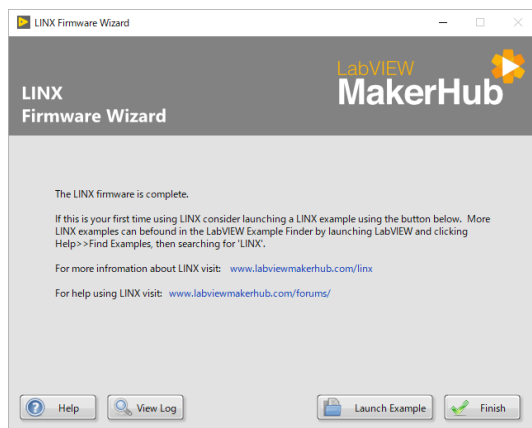


図 5-17 書き込み完了

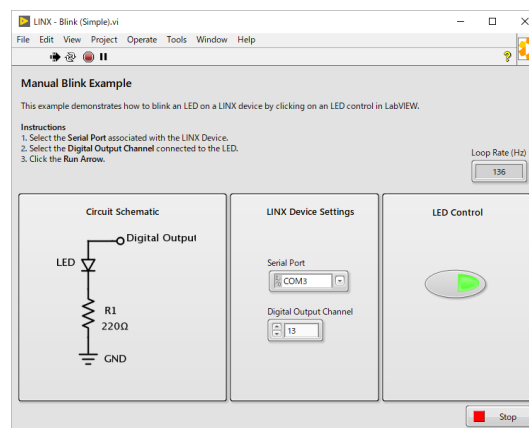


図 5-18 LINX - Blink (Simple).vi

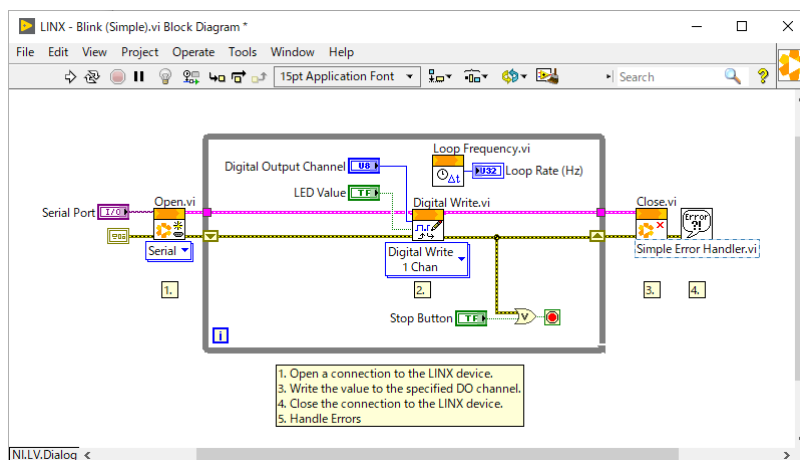


図 5-19 LINX - Blink (Simple).viのブロックダイアグラム

タン（矢印アイコン）を押してください。5秒ぐらい反応しませんが、Loop Rate(Hz)が100ぐらいを表示して動作開始したことがわかります（図5-18）。「LED Control」ボタンを押すとArduino UNOのランプが点灯あるいは消灯します。これで、LabVIEWでArduinoをコントロールする準備が整いました。ダイアグラムを見てください（図5-19）。デジタル出力のサンプルです。実行開始から数秒間反応しないのは**Open.vi**の中でボードに書き込まれたLINXファームウェアをチェックしているためです。ボードによってコマンドや入出力チャンネルが異なるためチェックに時間がかかっているようです。

## 5.4 スイッチカウンターを作る

配線をしますのでArduinoはUSBポートから抜いてください。デジタル入力のプログラムを作ります。ボタンを押すとカウンターが1ずつ増えて、10を超えると0に戻るカウンターです。

図5-20のようにブレッドボード上にタクトスイッチと抵抗を配置してください。タクトスイッチは同じ面から出ている2本の足がスイッチを押すと導通します。ブレッドボードの中には多数のレールが入っています。裏面のテープを剥がすと写真5-1のように金属レールが見えます。穴から入ってきた部品の足をレールのバネ部分で挟んで電気的な接続を行います。ブレッドボードは半田ごてを使わずに回路を作れるので試行錯誤が何度でもできてとても便利です。ブレッドボードには多くの種類があるので、初めて使うときには内部のレールの配置を確認してください。Arduinoの5Vピンからタクトスイッチの左端子、タクトスイッチの右端子から100オームの抵抗を通してグラウンドに至ります。タクトスイッチの右端子と抵抗が接続されているレールに8ピンからワイヤーを接続します。写真5-2とも比べてみてください。

デジタル入力のサンプルVI **LINX - Digital Read 1 Channel.vi**を使います。図5-21のようにヘルプメニューの「Find Examples..」を選択します。「NI Example Finder」が開きますので、**LINX - Digital Read 1 Channel.vi**をダブル

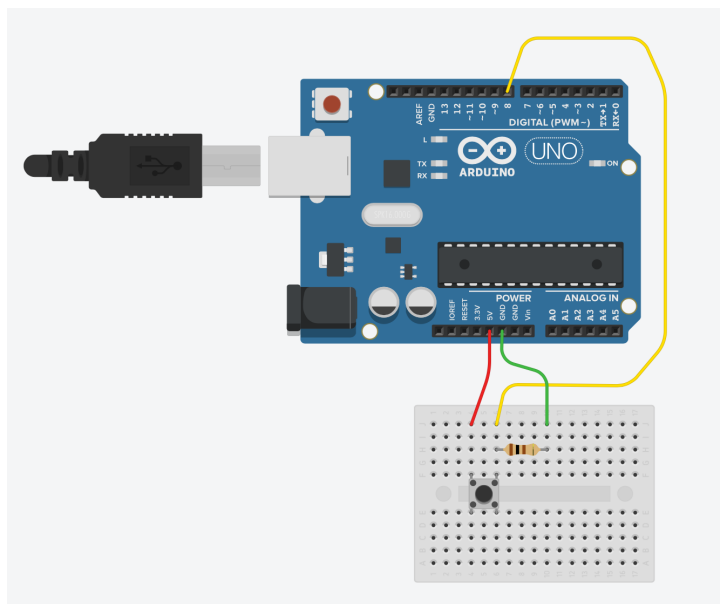


図 5-20 配線図

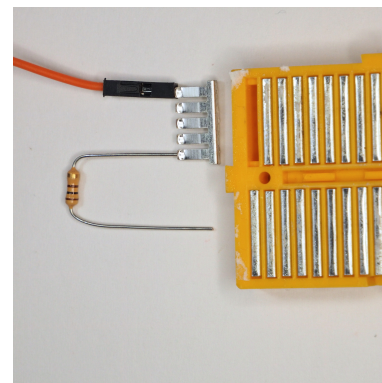


写真 5-1 ブレッドボードの構造

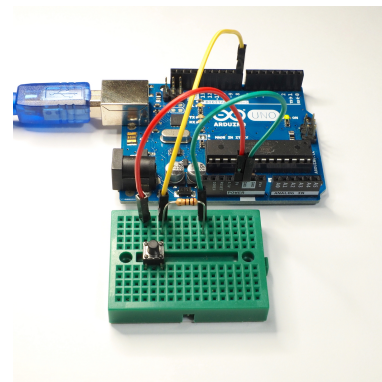


写真 5-2 部品の配置と配線

ルをクリックします (図5-22)。図5-23のようにArduinoが接続されているシリアルポートを選択し、DI Channelを「8」に設定して実行ボタンを押します。タクトスイッチを押すとDI Valuesが「ON」になることを確認してください。ブロックダイアグラム (図5-24) はLINX - Blink (Simple).viと似ています。

**PushCounter.vi**という名前で保存し、タクトスイッチが押されて離されるとカウントアップし、10を超えると0に戻るカウンターを作成しましょう。

ヒント1: スイッチがONになったらOFFになるまで待ちます。

ヒント2: シフトレジスタを使ってカウント値を保持します。

ヒント3: カウント値が10を超えた時はカウント値に0を入力します。

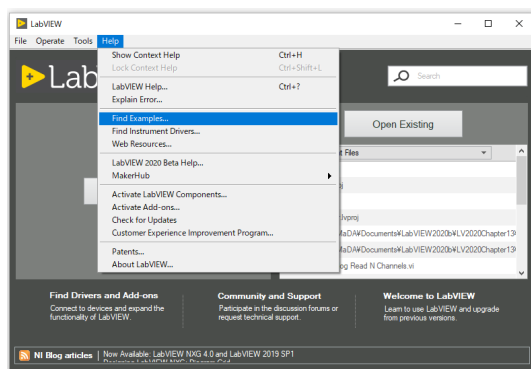


図 5-21 HelpからFind Examples...

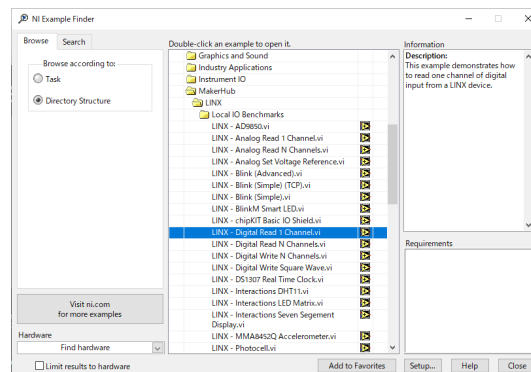


図 5-22 Find Examples...

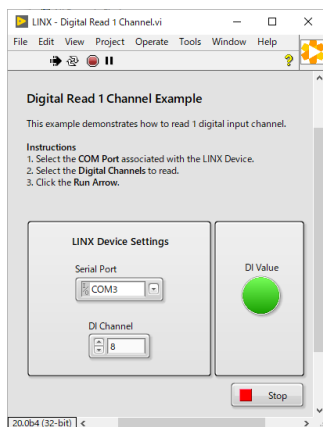


図 5-23 LINX - Digital Read 1 Channel.vi

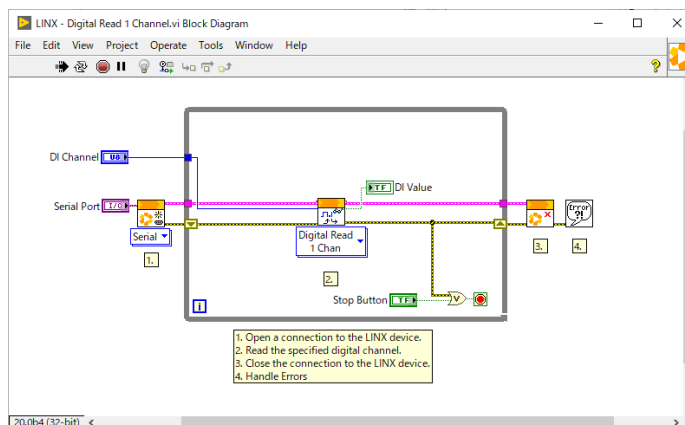


図 5-24 ブロックダイアグラム



## 5.5 ファンコントローラを作る

ArduinoをUSBポートから抜いて、**図5-25**のようにファンの配線を追加します。ファンの黒いワイヤーがグラウンド、赤いワイヤーを9ピンと接続してください。写真5-3も参考にしてください。

「NI Example Finder」を使ってLINXのサンプルVI **LINX - PWM 1 Channel.vi**を開いてください。ArduinoをUSBポートに接続し、**LINX - PWM 1 Channel.vi**のシリアルポートを設定します。PWMチャンネルは9に変更してください（**図5-26**）。実行ボタンを押して、「Duty Cycle (0-1)」に「0.5」を入力してください。ファンが回り始めたと思います。Duty Cycleが「0.0」から「0.4」ぐらいまでは回転しないかもしれませんが、Arduinoの異常ではありませんので心配しなくて大丈夫です。

「Duty Cycle (0-1)」のように制御器の名前に入力範囲やデフォルト値を入れるのは良い習慣です。PWMなので0-255と思い込んで、1より大きな数値を入力してしまうことを避ける効果があります。ただし、増分/減分ボタンを押すと「1.5」や「-0.5」になってしまうのは使いにくいですし、危険な場合があります。そのような場合はプロパティの「Data Entry」タブ（**図5-27**）で下限、上限、増分などを**図5-28**のように変更するようにしてください。

ブロックダイアグラムは**図5-29**のようになっています。**PWM Set Duty Cycle.vi**に0から1までの数値を入れれば良いことが分かりました。

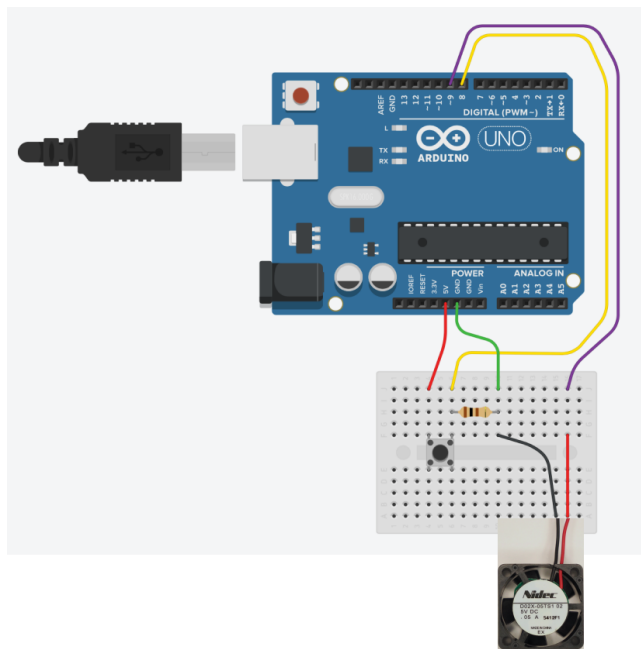


図 5-25 ファンの配線を追加

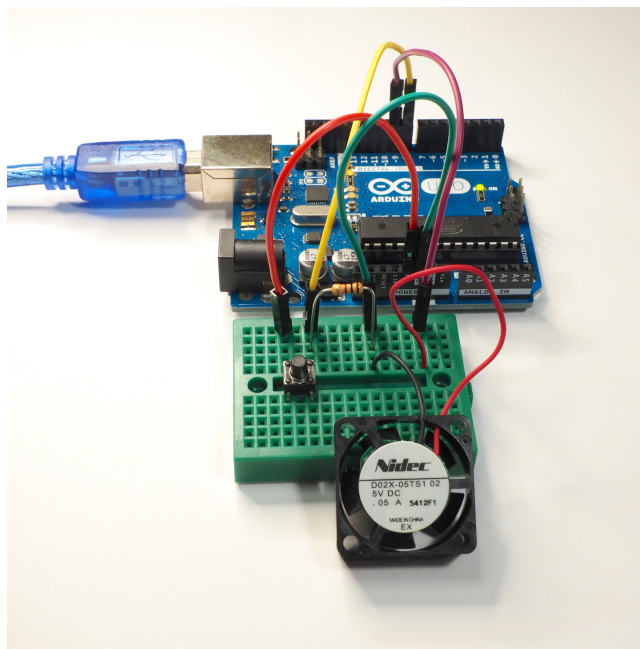


写真 5-3 部品の配置と配線



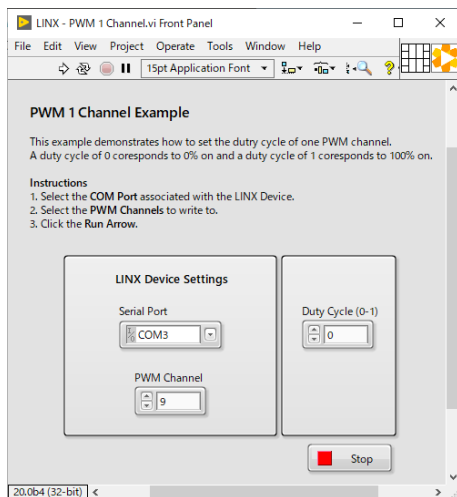


図 5-26 LINX - PWM 1 Channel.vi

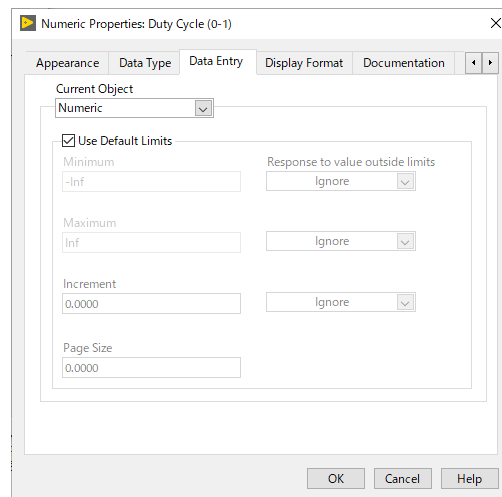


図 5-27 数値制御器のプロパティ

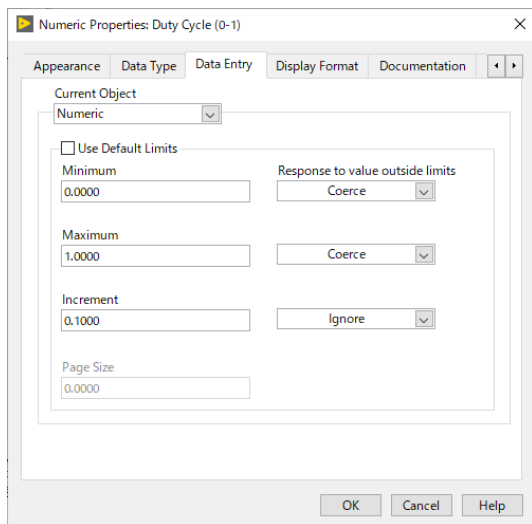


図 5-28 下限、上限、増分の設定

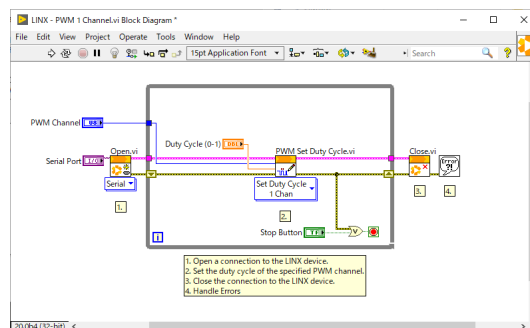


図 5-29 LINX - PWM 1 Channel.vi

さて、タクトスイッチでカウントアップする**PushCounter.vi**を改造してファンの風量をタクトスイッチでコントロールします。**PushCounter.vi**は図5-30のようなダイアグラムになっていると思います。

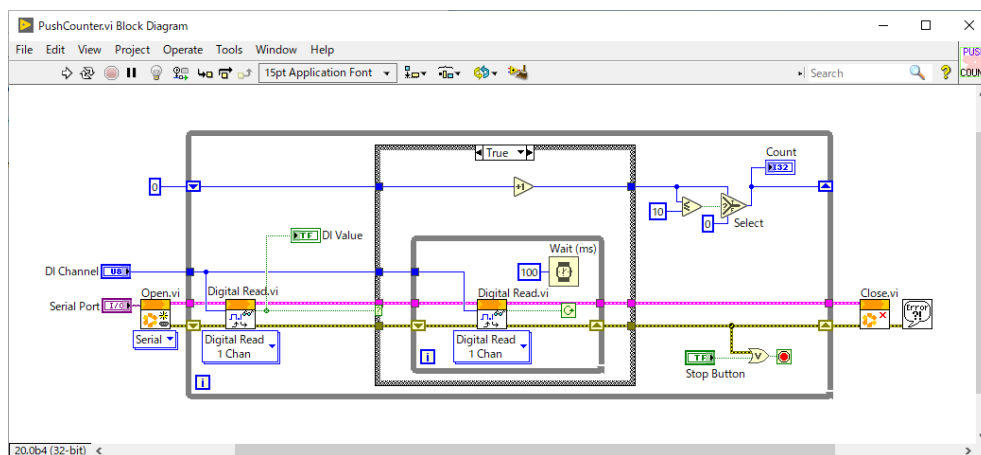


図 5-30 PushCounter.vi

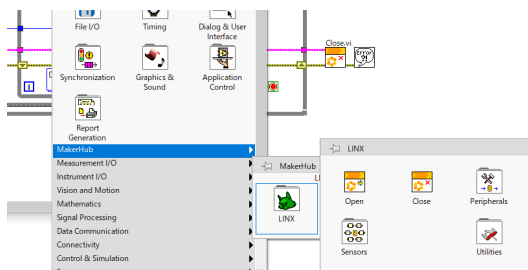


図 5-31 LINXの関数パレット

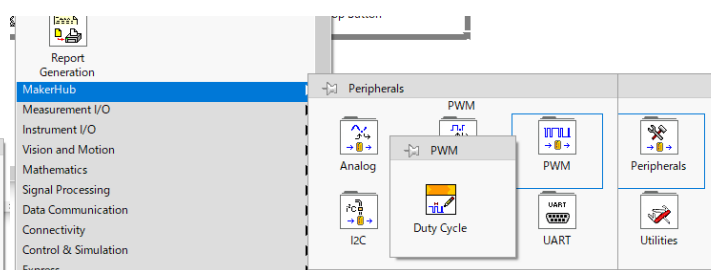
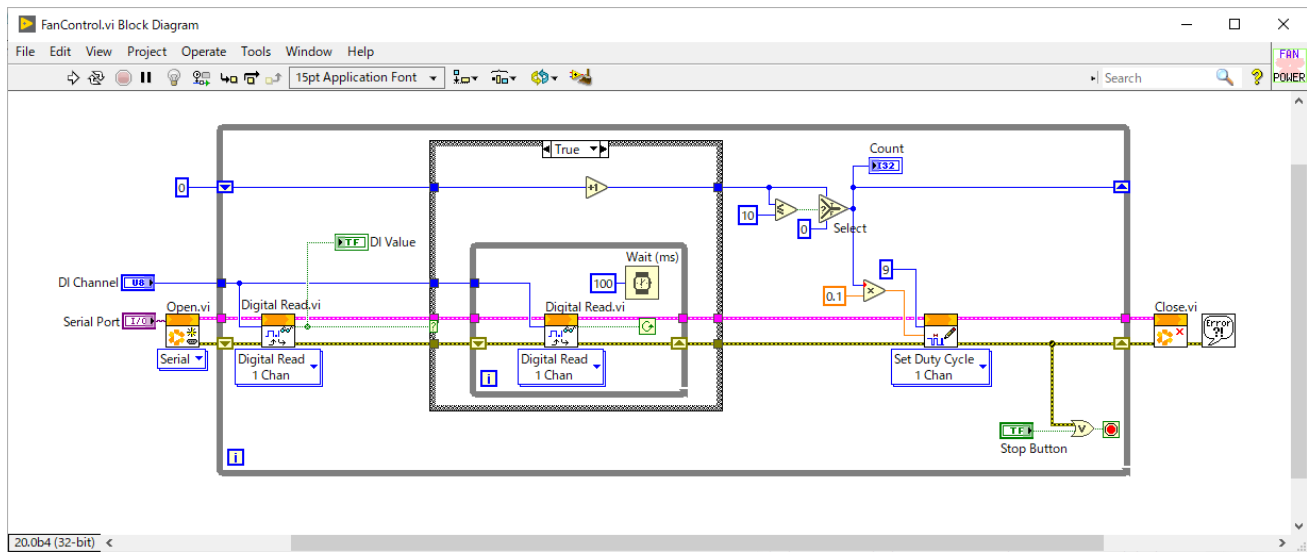


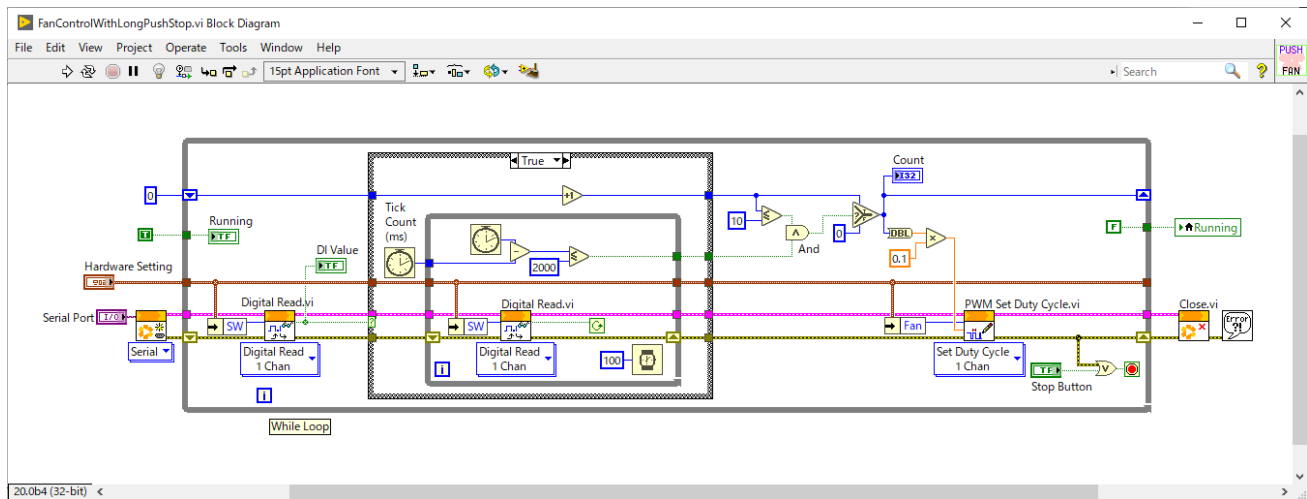
図 5-32 PWM Set Duty Cycle.vi

カウント値に0.1をかけて**PWM Set Duty Cycle.vi**の入力に接続すれば良いでしょう。LINXの関数パレットは「MakerHub > LINX」です(図5-31)。デジタル入出力やアナログ入力、PWMは「Peripherals」フォルダに入っています。**PWM Set Duty Cycle.vi**は図5-32のようにたどります。

とりあえず図5-33のようになったと思います。カウンターが大きくなると回転数が上がり、「0」で停止します。良い感じといえば良い感じなのですが停止させるのが面倒です。「10」までボタンを押した後でないと停止できません。タクトスイッチを2秒間長押ししたときに停止するようにしたプログラムが図5-34の**FanControlWithLongPushStop.vi**で



5-33 FanControl.vi



5-34 FanControlWithLongPushStop.vi

す。Tick Count (ms)でタクトスイッチが押されてから離されるまでの経過時間を測定しています。2秒以下であれば**True**を出力します。

（「10以下」 **And** 「2秒以下」）の時はカウント値をそのまま使いますが、それ以外のときにはカウント値を「0」にします。タクトスイッチとファンのチャンネル番号を「Hardware Setting」というクラスターにして配線をシンプルにしています（図5-35）。

**FanControlWithLongPushStop.vi**はタクトスイッチを2秒間長押しして「離れたときに」停止するプログラムなので、「タクトスイッチを2秒間長押しされたらタクトスイッチを離さなくても停止するプログラム」に改造してみましょう。

**FanControlWithLongPushImmediateStop.vi**（図5-36）はファンが停止するので一見良さそうなのですが、「Count」が「1」になります。**PWM Set Duty Cycle.vi**の実行が終わった後で、**Digital Read.vi**が**False**になるまでWhileループで待つ処理を入れれば正しく動作するようになりますので、さらに改造してみてください。

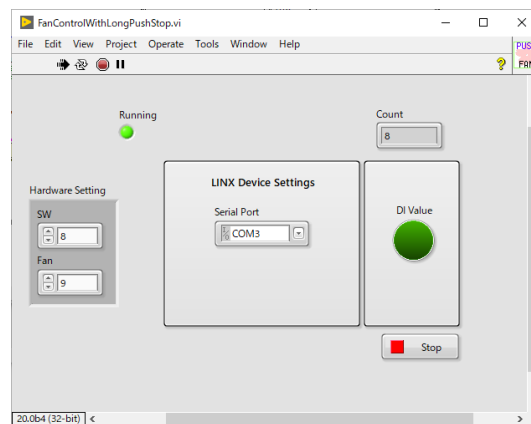


図 5-35 FanControlWithLongPushStop.viのフロントパネル

最後に注意事項です。LINX関数は**LINX Resource**というクラスターが入出力端子の一番上にあります。LINXは、Arduinoにコマンドを送り応答を受けて動作するのでLINX Resourceのワイヤーを分岐して2個以上のLINX関数に接続し

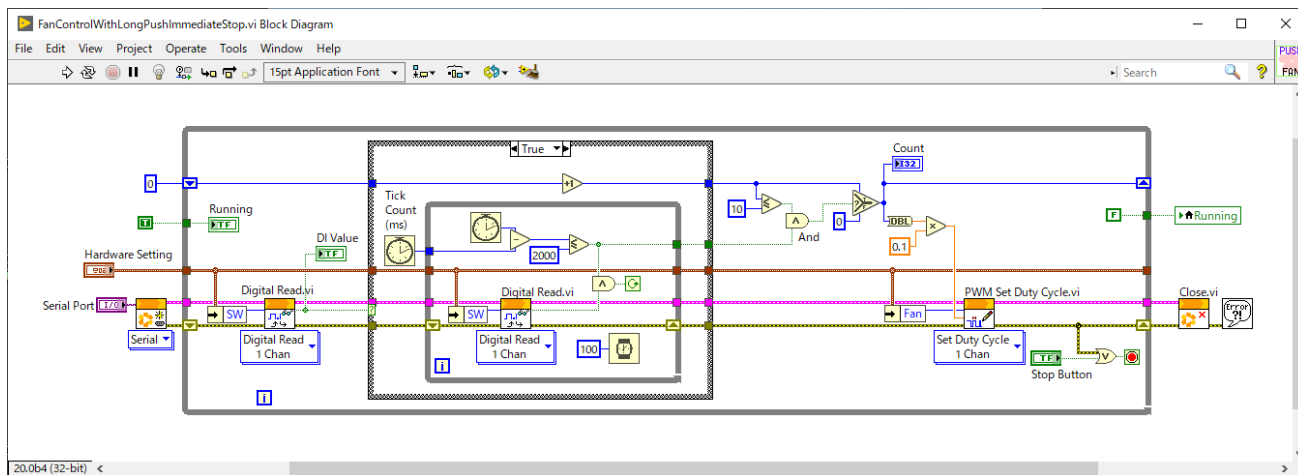


図 5-36 FanControlWithLongPushImmediateStop.vi

てはいけません。エラーにならない場合があるかもしれませんが、LINX Resourceを数珠つなぎするように関数を配置してください。

## 5.6 スイッチの動作を変えよう

FanControlWithLongPushImmediateStop.viの改造はどうでしたか？ ブロックダイアグラムが画面からはみ出していませんか？

ところで、LabVIEWのブール制御器は図5-37のように6種類のメカニカルアクションが選択できます。

「5.4 スイッチカウンターを作る」で考えていただいた PushCounter.viは「タクトスイッチが押されて離されるとカウントアップ」する動作でした。6種類のメカニカルアクションの中で、「Latch When Released（放されたらラッチ）」に近い動作です。「Latch When Released（放されたらラッチ）」のスイッチ動作をプログラムの的に実現したプログラムを紹介します。Push Counter(LatchWhenReleased).vi（図5-38）とFan Control with Long Push Stop(LatchWhenReleased).vi（図5-39, 図5-40）はプロがささっと作るとこんな感じというダイアグラムですから、ぜひ読み解いてください。

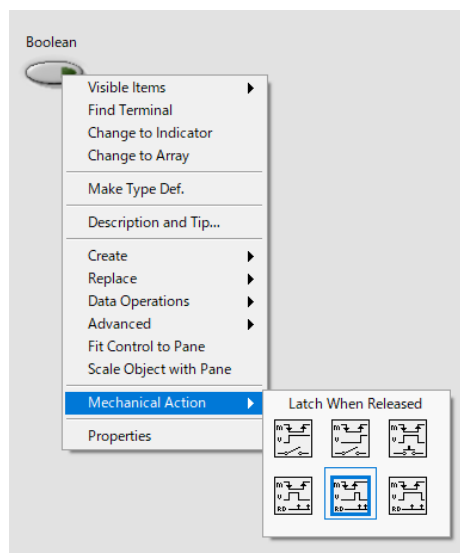


図 5-37 ブールスイッチの機械的動作

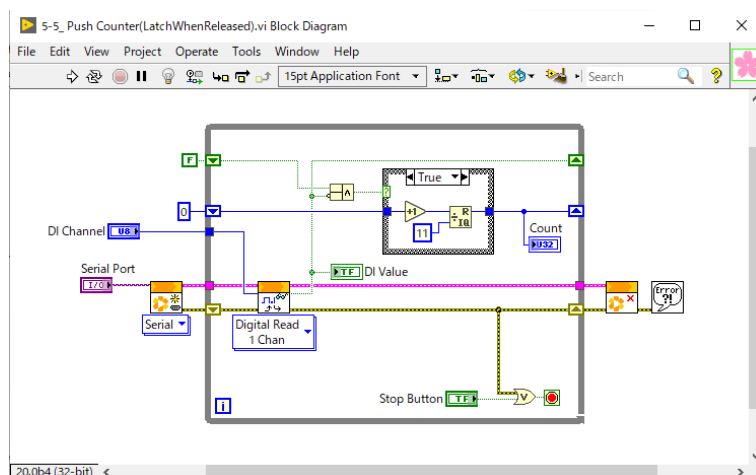


図 5-38 放されたらラッチのスイッチ動作

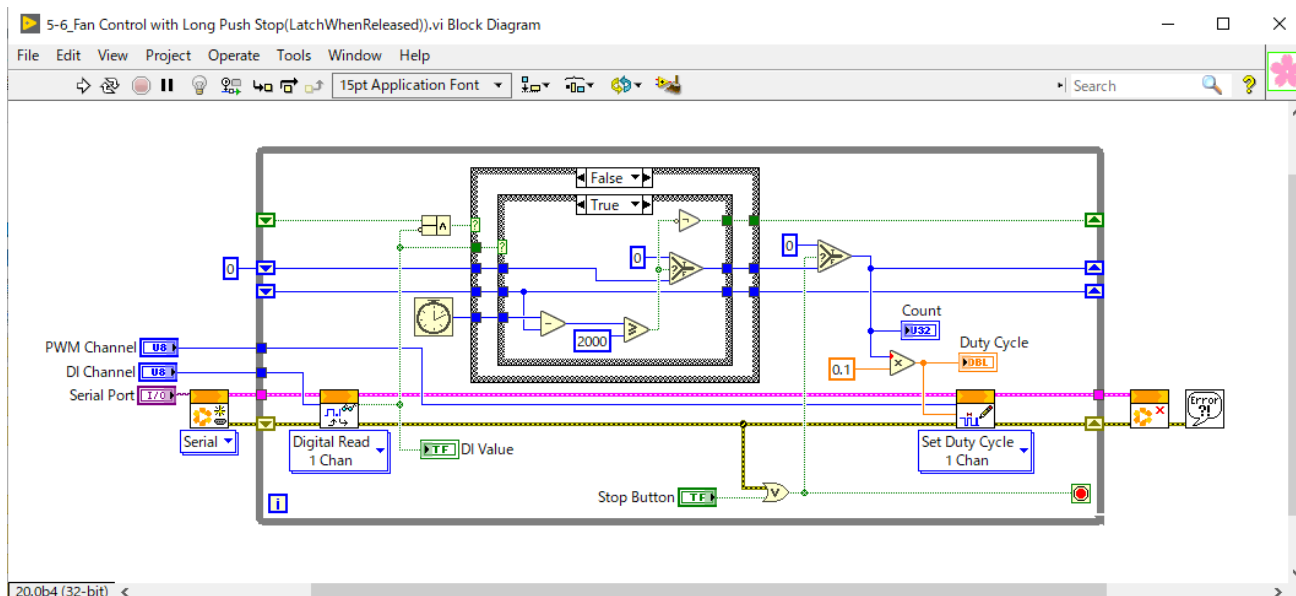


図 5-39 放されたらラッチのスイッチ動作で長押しで停止 (False & True Case)

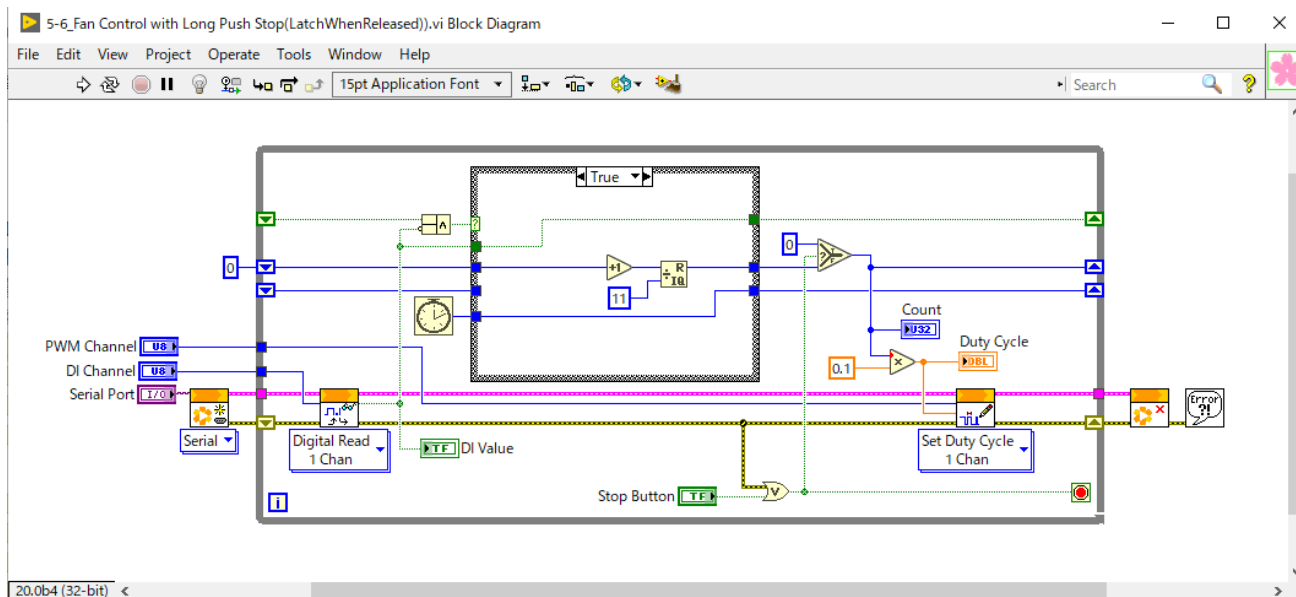
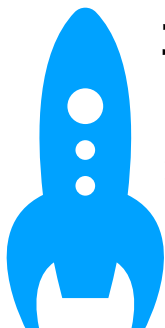


図 5-40 放されたらラッチのスイッチ動作で長押しで停止 (True Case)



## コラム5 LabVIEW NXGとハードウェア

Data Acquisition (DAQと略します)ハードウェアとは、電圧を測定したりデジタル信号を入出力したりできるハードウェアです。USBなどを使ってパソコンとつないで使います。LabVIEW NXGでは、DAQを使ったプログラミングに大きな変化があります。DAQを使うためには電圧の最大最小範囲や、1秒間に何回電圧を測るかといった設定が必要で、これらの確認はプログラミングがある程度終わってからということがほとんどでした。

LabVIEW NXGではプログラミングを行わずにDAQの設定と動作検証が行えるようになっています(図C5-1)。さらにここで行った設定から、自動的に制御コードを生成して、プログラミングの中で使うことができるようになります。

元々LabVIEWは、プログラミングを生業としないエンジニアでも計測や制御のプログラムを作れるようにという想いで開発されました。LabVIEW NXGではその想いを強く引き継いで、プログラミングをしなくてもできることの割合がLabVIEWに比べて多くなっています。

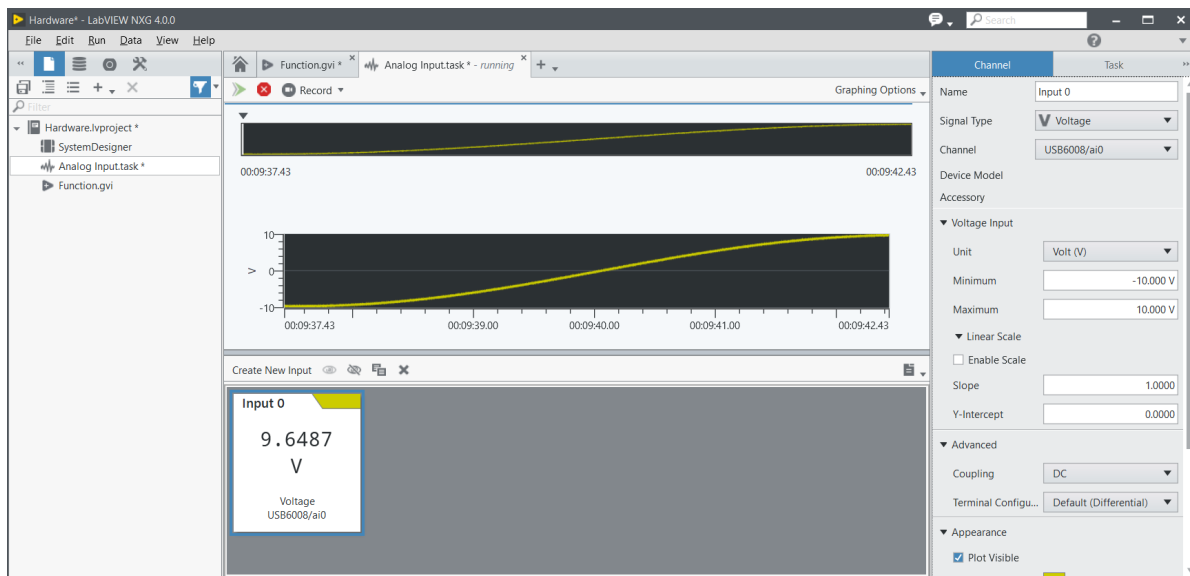


図 C5-1 LabVIEW NXGのHardware (ハードウェア) プロジェクト画面





# 第 6 章

## LEDの特性を調べてみよう



Arduinoのアナログ入力を3チャンネル使ってLEDの電圧電流特性を測定します(写真6-1)。測定データを理論式に当てはめて回帰曲線を求めます。少し難しい式ですが、データ解析の例として参考になると思います。

**[キーワード]** X-Yグラフ、1D配列ソート、対数変換、直線回帰

**[使用する部品]** Arduino UNO 1個、ブレッドボード 1個、LED 1個、抵抗(100Ω) 1個、可変抵抗(10kΩ) 1個、ワイヤ 6本

### 6.1 LEDの電圧電流特性の実験回路の組み立て

LEDを点灯する時には電流が流れ過ぎないように電流制限抵抗を入れる必要があります。LEDは電圧が順方向電圧(V<sub>f</sub>)を超えると指数関数的に電流が増加し光量が大きく変化します。電圧で光量を調整するよりも、抵抗を入れて回路に流れる電流を調整した方が簡単なのです。この章では指数関数で急激に電流が増加する様子を実験的に見てみようと思います。

実験にはLED、100Ωの固定抵抗、10kΩの可変抵抗、ワイヤーが6本、ブレッドボードとArduino UNOが必要です。なお、章末の

**6.Appendix** に可変抵抗器や実験回路の補足説明を用意しました。

ArduinoをPCのUSBポートから外して、図6-1のように配線してください。

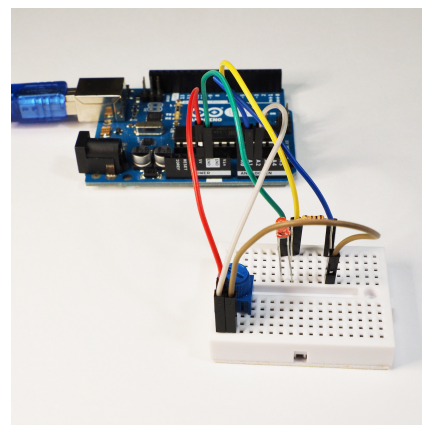


写真 6-1 LEDの電圧電流特性の測定

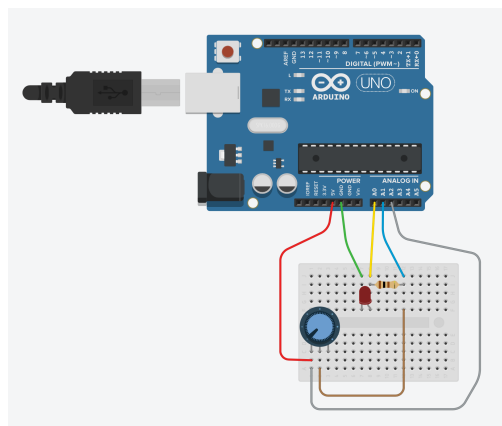


図 6-1 実験回路の配線図

Arduinoの5Vピンから10kΩの可変抵抗と100Ωの固定抵抗とLEDを直列に配線し、グラウンドに接続します。可変抵抗は3本足です。つまみを左いっぱいにしてつまみの矢印の先が左下になるように置いた時に、上から見て左端を5V、中央を100Ωの固定抵抗に接続してください。LEDは極性があるので足が短い方をグラウンドに接続してください。

電流は5V→可変抵抗（0kΩ～10kΩ）  
→100Ω→LED→グラウンドへと流れます。可変抵抗の値を変えるとLEDに加わる電圧と流れる電流が変化します。その電流と電圧をArduinoのアナログ入力で測定するのです。

100Ωの固定抵抗とLEDの接続ポイントからワイヤーを出して「A0」に接続します。これはLEDに加わる電圧です。10kΩの可変抵抗と100Ωの固定抵抗の接続ポイントからワイヤーを出してA1に接続します。「A1」の電圧からA0の電圧を引くと100Ωの抵抗に加わる電圧です。オームの法則（ $I=V/R$ ）で回路を流れる電流、つまりLEDを流れる電流がわかります。5V電源が安定していることを見るために、5V電源と10kΩの可変抵抗の接続ポイントからワイヤーを出して「A2」に接続します。

それではもう一度配線を確認して、OKならばArduinoをPCのUSBポートに接続してください。USB端子に接続するとLEDは光り始め、可変抵抗を右に回すと暗くなります。これで実験回路の準備ができました。

## 6.2 LEDの電圧と電流の測定

Arduinoのアナログ入力を3チャンネル使用してLINXでLEDの電圧電流特性を測定します。

NI エグザンプルファインダ（図6-2）でLINX - Analog Read N Channels.viを開き、プロ

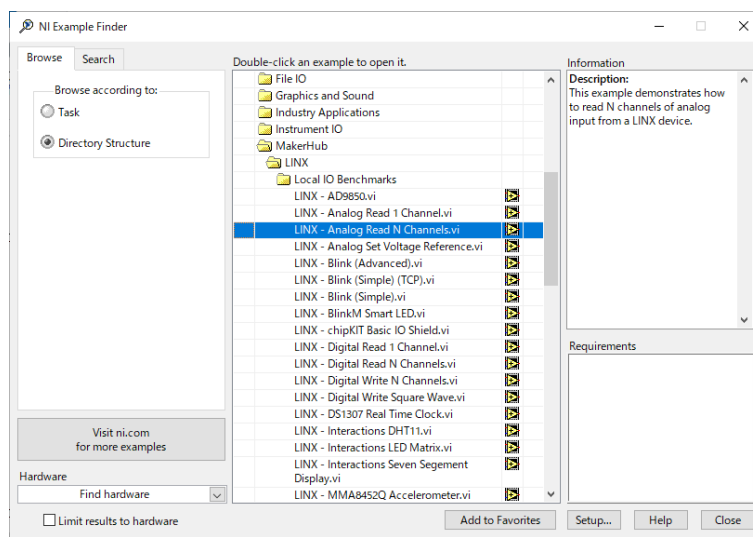


図 6-2 NIエグザンプルファインダ

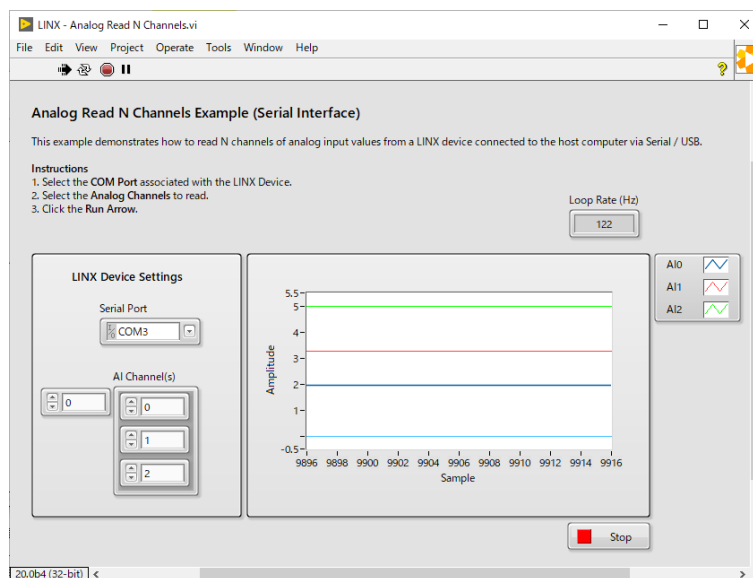


図 6-3 LINX - Analog Read N Channels.vi

グラムを作成しているフォルダに保存します。Serial Port（シリアルポート）制御器の右端の「▽」マークを左クリックして表示されるメニューからArduinoが接続されているCOM番号を選択します。ArduinoをUSBポートに接続しているのにメニューに表示されていない場合はメニューの一番下にある「Refresh」を選択するとメニューが更新されます。AI Channel(s)配列の要素にアナログ入力の番号を入力します。「A0」から「A2」ですから「0」、「1」、「2」です。実行ボタンを押して可変抵抗を右左に回すと波形チャート(Waveform Chart)「Analog Data」の表示が変化します(図6-3)。「A0」が2V付近、「A1」は可変抵抗の位置によって2V付近から5Vの間、「A2」は5Vです。

0V付近にもデータが表示されていることに気がついた方もいるかもしれません。波形チャートの右にある「Plot Legend」にマウスカーソルを移動し、表示される小さな水色の四角形を下にドラッグすると「Plot 3」です。図6-4のように配列「AI Channel(s)」の枠を下に広げてみても入力されている要素は3個で、3チャンネルしか設定していません。

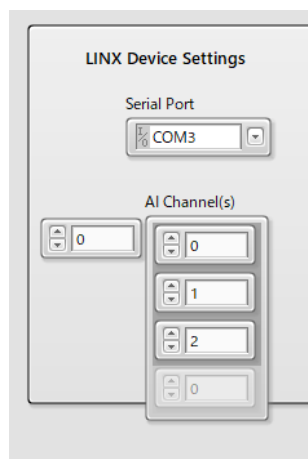


図 6-4 チャンネル配列の要素

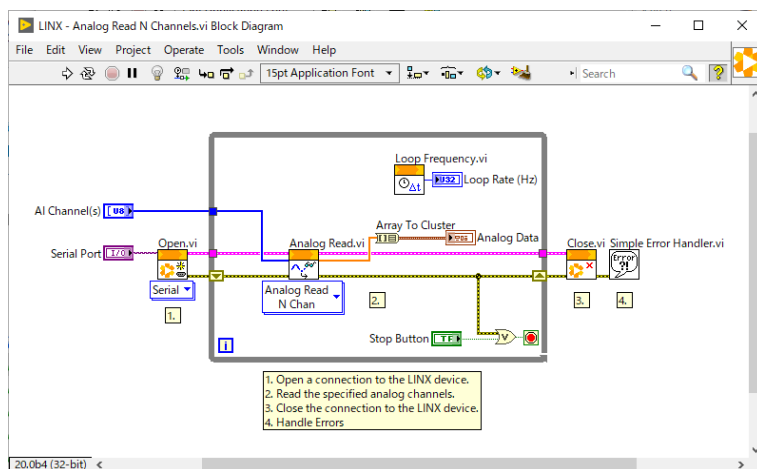


図 6-5 ブロックダイアグラム

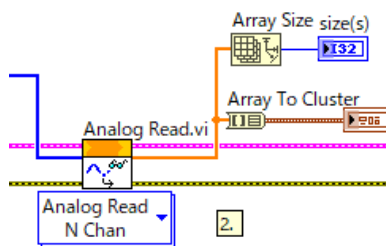


図 6-6 出力配列のサイズ

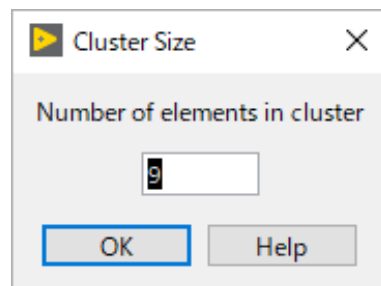


図 6-7 クラスタの要素数

「Window」メニューの「Show Block Diagram」でブロックダイアグラム（図6-5）を表示させて、ブロックダイアグラムを右クリックして関数パレットを開き、Arrayパレットにカーソルを合わせます。

「Array Size」を左クリックして**Analog Read.vi**の近くに配置し図6-6のように配線し、配列の要素数を調べてみましょう。実行ボタンを押して「size(s)」表示器を見ると期待通り3です。**Analog Read.vi**と波形チャート間にある**Array To Cluster**をダブルクリックすると「Cluster Size」と言う小さなウィンドウ（図6-7）が表示されます。3を入力してOKボタンを押してください。これで波形チャートの4本目の線は消えたと思います。**波形チャートに複数のプロットを表示するにはクラスターにまとめて接続する必要があることもヘルプ画面で確認してください。**

**LINX - Analog Read N Channels.vi**を**LED VI Curve.vi**として別名で保存します。**LINX - Analog Read N Channels.vi**の説明書きがフロントパネルに残っています。不要になりましたので削除したいのですが、「lock」されているので削除できません。説明書きの周辺をマウスで左クリックしながらドラッグするとテキストを選択することができます。ツールの右端のアイコンの「Reorder」ツールで「unlock」を選択すると削除できるようになります（図6-8）。

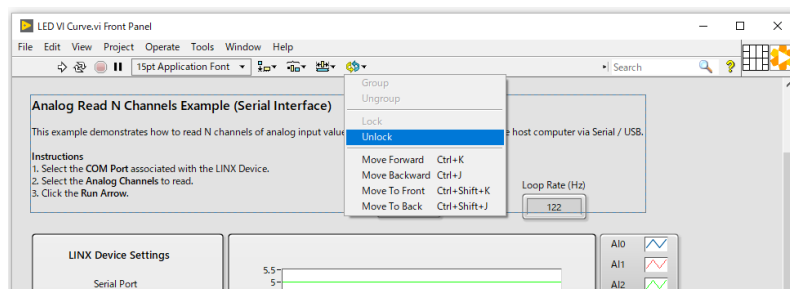


図 6-8 説明テキストのロック解除

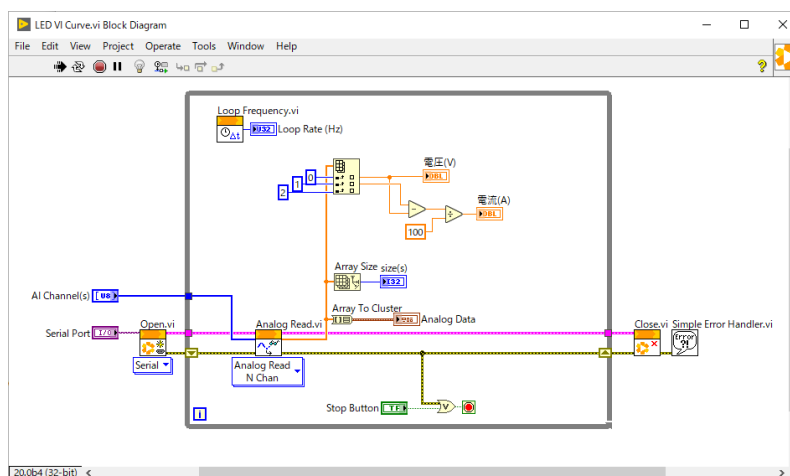


図 6-9 電圧と電流を表示するブロックダイアグラム

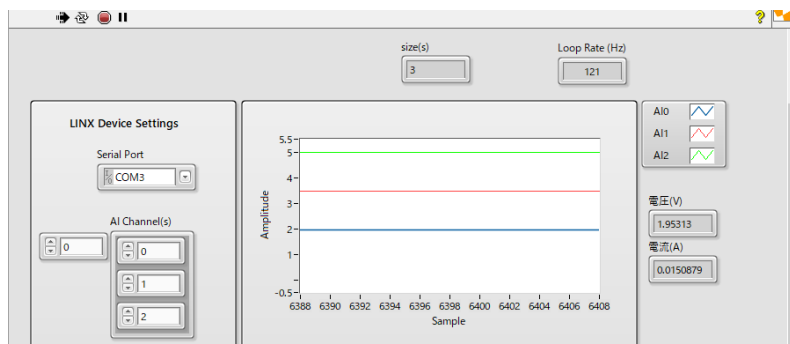


図 6-10 電圧と電流が表示されたフロントパネル

Analog Read.viから出力された配列からIndex Array関数を使って要素0、要素1、要素2を取り出します。要素0がLEDにかかる電圧です。要素1から要素0を引いた値が100Ωの抵抗にかかる電圧で、抵抗値100Ωで割ればこの回路を流れる電流となります（図6-9）。こうしてLEDにかかる電圧と流れる電流を求めることができます。LEDにかかる電圧と流れる電流に数値表示器を接続して、電圧が高くなり電流が大きくなると明るく光り、電圧が低くなり電流が小さくなると暗くなることを確認してください（図6-10）。

## 6.3 LEDの電圧電流特性曲線の表示プログラム

横軸を電圧、縦軸を電流としてX-Yグラフに表示してみましょう。もう自分でプログラムを作ることができるかもしれませんが、サンプルプログラムを用意しました。プログラムフォルダの6-2\_LED VI Curve2.viを開いてください。データ追加ボタンが押されると測定した電圧と電流がそれぞれの配列に追加され、電圧の配列と電流の配列をX-Yグラフに表示します。

「Serial Port」とAI Channel(s)を設定してから実行ボタンを押します。可変抵抗で電圧と電流を変えて「データ追加ボタン」を押してください。どんどんデータを追加していくと右肩上がりの曲線が見えてきます（図6-11）。まばらなところをできるだけ埋めるように可変抵抗と「データ追加」ボタンを操作してください。

「回帰分析」というボタンについては 6.4 LEDのV-I特性データの回帰分析で詳しく説明します。

なお、6-2\_LED VI Curve2.viにはデータ保存機能がありませんので、プログラムを停止するとデータが使えなくなります。回帰分析の説明には電圧と電流のデータが必要ですので、まだプログラムは停止させないでください。

ブロックダイアグラムを開いて予想していたようなプログラムになっていたかどうか見てくだ

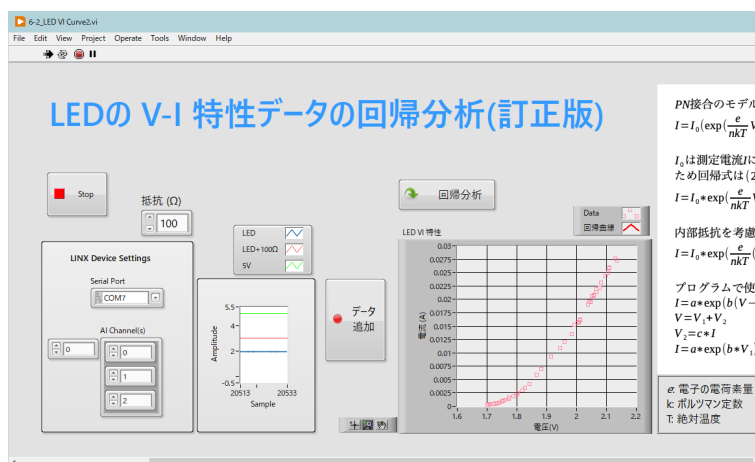


図 6-11 データをX-Yグラフに追加する

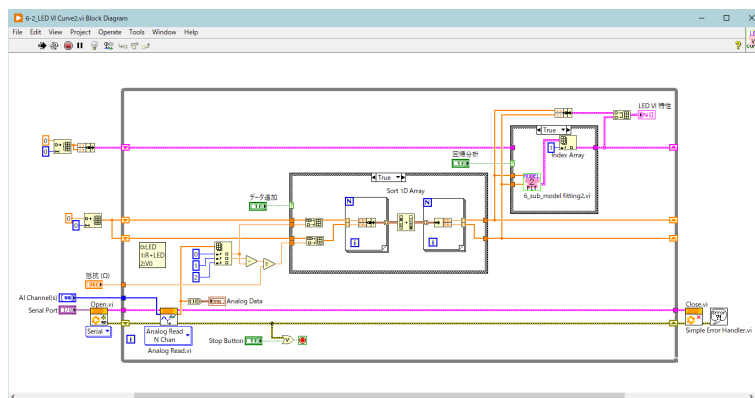


図 6-12 ブロックダイアグラム

さい (図6-12)。

「データ追加」ボタンが押された時に動作するケースストラクチャに注目してください。ここでは電圧が低いデータから高いデータに並べ換えています。電圧と電流のデータをペアにした状態で処理しないと意味がありませんので、少し込み入っています。電圧、電流ともに新しいデータを配列に追加した後で、電圧と電流のクラスターを要素にした1次元配列を作ります。**Sort 1D Array**でクラスターの1次元配列をソートします。クラスターの要素の0番をキーにしてソートされますので、電圧をクラスター要素0番にしていることに注意してください。ソートした後で、クラスターを分解して電圧、電流の配列に戻します。**Sort 2D Array**関数も使えますので気に入った方法を使ってください。X-Yグラフへの表示はXの1次元配列とYの1次元配列をクラスターにしてX-Yグラフに入力します。複数のプロットを表示する時には配列にしてからX-Yグラフに入力します。

## 6.4 LEDのV-I特性データの回帰分析

電圧を変えながらデータを追加すると電圧が高くなるにつれて電流が増えていく様子がよくわかります。同時に、データが意外にばらついていることにも気がつくと思います。実験データを取る時に、きちんと制御できない要因により誤差が生じます。可変抵抗が回した時の圧力で抵抗値がふらついているのかもしれませんが、LEDの温度変化によって特性が変化したため誤差が生まれる可能性もあります。また、Arduinoでの電圧と電流の読み取りは時間差があるため誤差の原因となります。

回帰分析を行うことにより、誤差を取り除いて特性を理解しやすい形で表現することができます。LEDはLight Emitting Diodeの頭文字を並べたもので、ダイオードの一種です。ダイオードのPN接合の電圧電流特性は図6-13の式(1)に書かれたモデル式で表すことができます。モデル式は現象を単純化して関係を数式に表したものです。式で使っている記号は図6-14にまとめました。見慣れない式かもしれませんが、高校2年生ぐらいの数学の授業で習う指数関数です。大まかに言えば、電圧Vを変化させると電流が指数関数で増加するという内容です。ほとんどのLEDでは逆方向飽和電流  $I_0$  は  $10^{-30}$  アンペア程度のとても小さな数値なので、式(2)のように簡略化することができます。

LEDに流れる電流が大きくなると内部抵抗 (r) による電圧降下が無視できなくなってきます。そのような状況を表したのが図6-15の式(3)です。(-r\*I) が内部抵抗による電圧降下です。プログラムで扱いやすいように係数をa, b, cに変更したものが式(4)です。式(5)ではLEDの内部でダイオードの接合部と内部抵抗が直列に接続されていて、ダ

$$I = I_0 (\exp(\frac{e}{nkT} V) - 1) \quad \dots (1)$$

$$I = I_0 * \exp(\frac{e}{nkT} V) \quad \dots (2)$$

図 6-13 ダイオードのモデル式

e : 電子の電荷素量  
k : ボルツマン定数  
T : 絶対温度  
n : 理想ダイオード因子  
 $I_0$  : 逆方向飽和電流

図 6-14 記号

$$I = I_0 * \exp(\frac{e}{nkT} (V - (r * I))) \quad \dots (3)$$

$$I = a * \exp(b(V - (c * I))) \quad \dots (4)$$

$$V = V_1 + V_2 \quad \dots (5)$$

$$V_2 = c * I \quad \dots (6)$$

$$I = a * \exp(b * V_1) \quad \dots (7)$$

図 6-15 内部抵抗を考慮したモデル式

ダイオード接合部の電圧 $V_1$ と内部抵抗の電圧 $V_2$ に分圧されていると仮定したことを示しています。直列接続なので、ダイオード接合部も内部抵抗も流れる電流 $I$ は同じ値になります。**式(6)**は内部抵抗の電圧と電流が原点を通る1次式であることを示しています。**式(7)**はダイオード接合部の電圧と電流の関係が指数関数であることを示しています。もしも、ダイオードの接合部と内部抵抗がつながっている部分の電圧をテスターで測定できるなら、全く簡単なことなのですが、ダイオードと抵抗では電圧と電流の関係が異なることを利用して $V_1$ と $V_2$ を推定します。

「回帰分析」ボタンを押すとX-Yグラフ上に回帰曲線が表示されます(図6-16)。実線で描かれた曲線が、多数のデータ点から回帰された回帰曲線です。

回帰分析を行っているサブVIを開いてみましょう。

**6-2\_LED VI Curve2.vi**のブロックダイアグラムを表示して、「回帰分析」ボタンがTrueの時に選択されるケースストラクチャの中にある**6\_sub model fitting2.vi**のアイコンをダブルクリックします。図6-17のようにフロントパネルが表示されますが、制御器や表示器には値が表示されていないと思います。**6-2\_LED VI Curve2.vi**のフロントパネルを前面に出して、「回帰分析」ボタンをもう一度押してみてください。制御器に値が入力されて、グラフにデータが表示されます

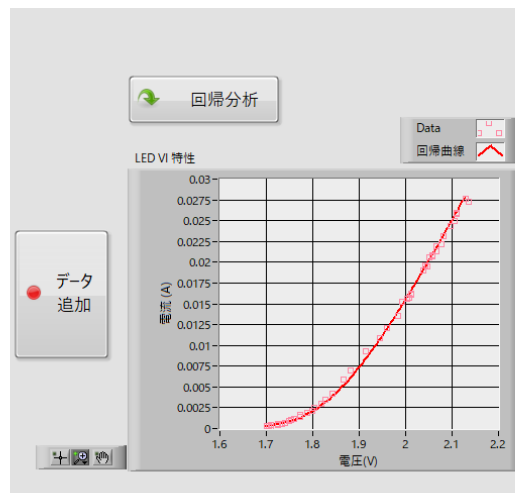


図 6-16 回帰曲線の表示

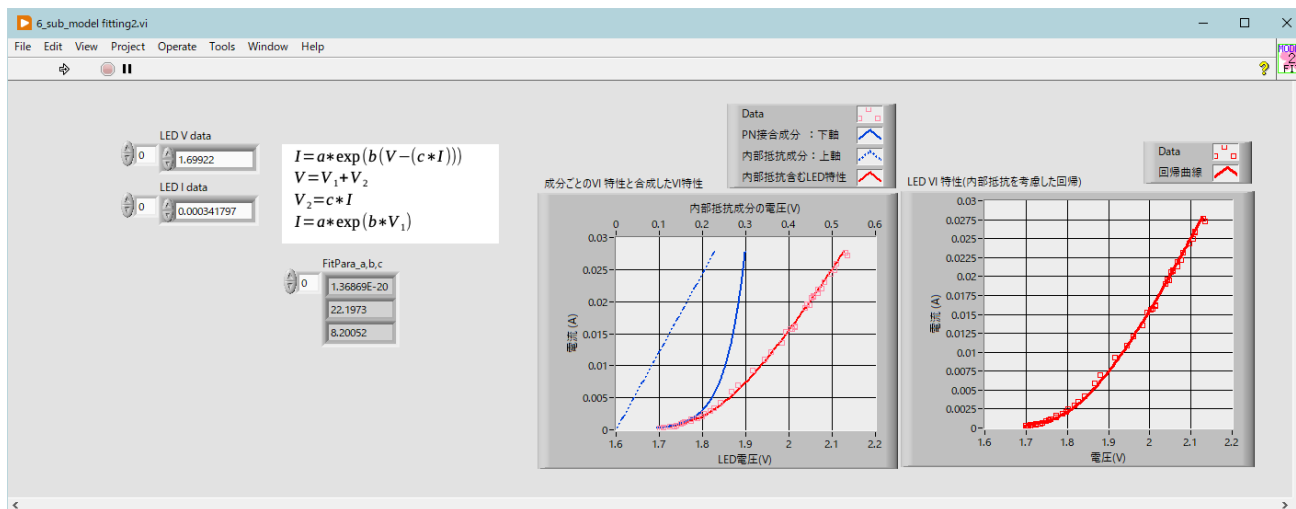


図 6-17 回帰分析を行うサブVI



す。右のグラフが最終的な回帰結果で、**6-2\_LED VI Curve2.vi**のフロントパネルのグラフに表示されているグラフと同じものです。左のグラフはダイオード接合部の電圧 $V_1$ と電流 $I$ が青の実線、内部抵抗の電圧 $V_2$ と電流 $I$ が青の破線で示されています。同じ電流値同士で $V_1$ と $V_2$ を加算したのが赤い実線になります。

**6\_sub model fitting2.vi**のブロックダイアグラムを開くと図6-18のように表示されます。このブロックダイアグラムでは回帰計算を行う**6\_sub\_Fit2.vi**を使用しています。Whileループの中で条件を変えて回帰計算を行い、その中で一番誤差が少なかった条件を見つけ、その条件で最終的な回帰結果を得ています。

それでは、**6\_sub\_Fit2.vi**を開いてみましょう。ブロックダイアグラムは図6-19のように表示されます。電圧と電流の測定データをそれぞれ1次元配列として入力します。ダイオード接合部の特性を内部抵抗による電圧降下が少ない部分で回帰するために数値制御器**UpperTh**で分割する指標を指定します。1次元配列の分割は**Split 1D Array.vi**を使用します。**Exponential Fit.vi**は指数関数回帰するときに係数の範囲を指定することができます。ここでは回帰する式に合わせてoffsetを0にしました。また、amplitudeとdampingは0以上に限定しました。**Exponential Fit.vi**の出力のamplitudeが回帰式の係数a、dampingが回帰式の係数bに対応しますので、ダイオード接合部の関数を仮決めすることができました。これらの係数を使って、測定された電流データに対応する電圧 $V_1$ を算出することができます。測定した電圧データ $V$ から $V_1$ を引くことで内部抵抗にかかっている電圧 $V_2$ も算出します。このような計算は関数アイコンを組み合わせてもできますが、フォーミュラノードを使った方が見通しが良くなります。

次に**Linear Fit.vi**を使って内部抵抗を回帰します。電流測定データと $V_2$ を入力し、パラメータ境界の入力でinterceptを0に限定します。slope出力として抵抗値が得られるように、**X入力**が電流で、**Y入力**が電圧としていることに注意してください。これでパラメータa, b, cが仮決めできましたので、電流データに対して $V_1$ 、 $V_2$ を求めてLEDの電圧 $V$ を算出します。測定した電圧データと算出した電圧との誤差を計算して出力します。

ここで、もう一度**6\_sub model fitting2.vi**のブロックダイアグラムを見てみましょう。左のWhileループの中で測定データ配列の要素数からループ回数分減じた値を**6\_sub\_Fit2.vi**の数値制御器**UpperTh**に入力しています。

**6\_sub\_Fit2.vi**から出力されるrms valueは徐々に低下し、解析データが少なくなりすぎるとまた増加します。最初のrms値の5倍を超えたところを目安にループを停止して、rms値が最小だった**UpperTh**入力値を見つけ、最適な回帰結果としています。

最終的な回帰結果ではパラメータa, b, cが得られますので、パラメータaの値を確認してみましょう。パラメータaはモデル式(1)で使われている逆方向飽和電流 $I_0$ と同じですが、十分小さいのでしょうか？式(2)のように簡略化したことは妥当だったのか考えてみてください。

この章で行なったLEDの回帰分析は大学の実習で行われるような内容でしたので、説明不足で理解しづらい点もあったと思います。ブロックダイアグラムは読みやすいと思いますので役に立ちそうだと思う方は研究してみてください。



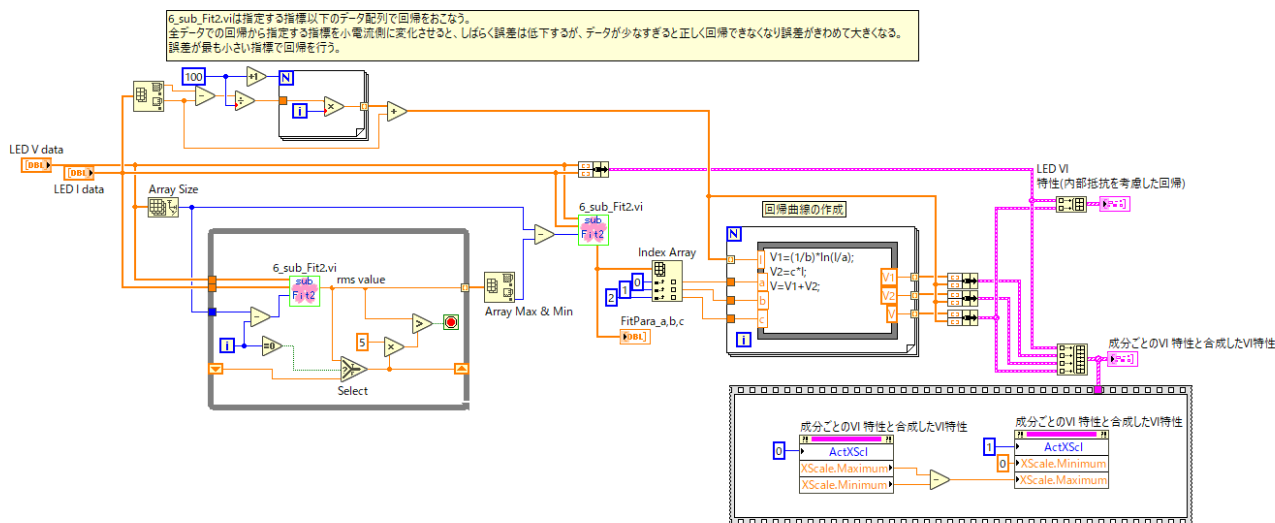


図 6-18 ブロックダイアグラム

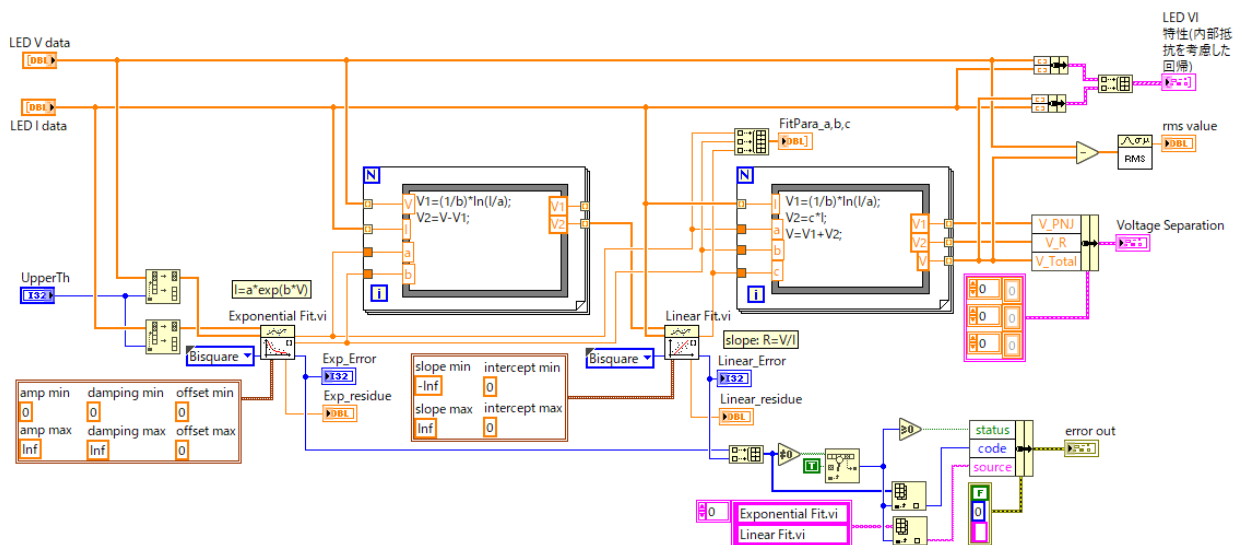


図 6-19 回帰分析を行うサブVI

## 6.Appendix 可変抵抗器や実験回路についての補足説明

### 6.Appendix.1 可変抵抗器について

読者の中には「みんなのArduino入門：基本キット」ではなく「Arduinoをはじめようキット」をお持ちの方もいるかもしれません。それには可変抵抗が入っていないので別途入手してください。半固定抵抗や可変ボリュームとも呼ばれ、 $10k\Omega$ であれば本文の写真と同じ形でなくてもかまいません。**写真6-A1**は小さなドライバーを使ってつまみを回すタイプです。**写真6-A2**の赤い①②③はピン番号で、中には書いていないものもありますが3本のリード線の位置関係で予想できます。

中身はほぼ同じで、分解すると**写真6-A3**のようになっています（外装が水色のものを破壊）。なお四角いものは減速ギアが入っていて片道25回転の精密調整できるタイプです。

1番ピンと3番ピンの間を渡っている黒い帯が抵抗体です。抵抗値 $10k\Omega$ というのは1-3間の値です。つまみを回すと金色のバネが2番ピンと抵抗体に接触しながら回ります。角度によって抵抗体への接触位置が変わり、1-2間と2-3間の抵抗値が連続して変わります。

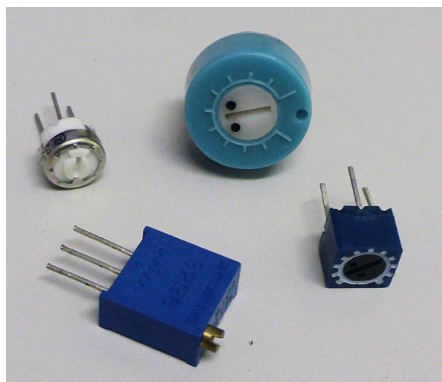


写真 6-A1 いろいろな可変抵抗器

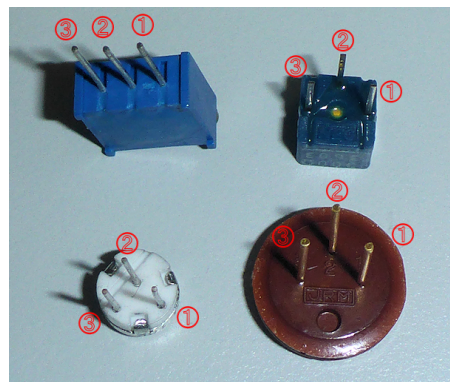


写真 6-A2 可変抵抗器のピン

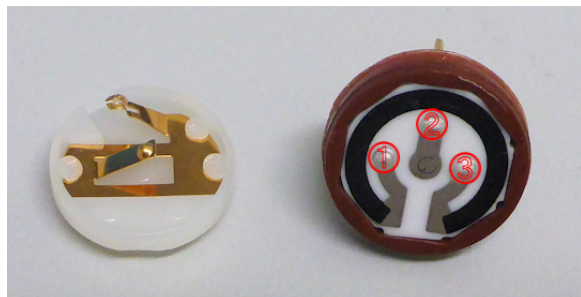


写真 6-A3 可変抵抗器の内部

## 6.Appendix.2 つまみの回転方向と抵抗値

一般的につまみを右に回し切ったときに1-2間が最大（ここでは10k $\Omega$ ）、2-3間が0 $\Omega$ になります。3端子の抵抗分圧回路として、または2端子の抵抗としても使えます。6章では直列の電流制限抵抗として使用するのので、2端子（1-2間または1-3間）を使います。

1-2または1-3のどちらを使うかによって、つまみを回す方向とLEDが明るくなる方向が入れ変わります。写真6-A4のうち左側は右回しで暗く、右側は右回しで明るくなります。予想と逆だったら差し替えればOKです。また逆のままでも測定には支障ありません。

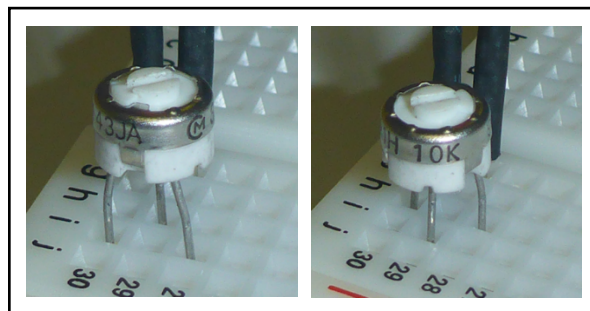


写真 6-A4 ブレッドボードへの挿し方

## 6.Appendix.3 抵抗値の表記

部品に書かれている抵抗値を読み取る時表記の混在に注意が必要です。写真6-A5は左から10k $\Omega$ と300 $\Omega$ です。これらは10Kや300 $\Omega$ と書かれているのでわかりやすいですが、右端は何 $\Omega$ だと思いますか？ ヒントは104です。

法則は「2桁の値+1桁の指数（10の累乗）」です。この場合 $10 \times 10^4$ （10の後に0が4個続くと覚えてもよい）で、 $100000 = 100\text{k}\Omega$ になります。例えば472は $4700 = 4.7\text{k}\Omega$ です。単位がなく1の桁が0以外ならこの表現とってください。単に100と書かれている場合は10 $\Omega$ または100 $\Omega$ の可能性がありますのでテスターなどで測って確認してください。

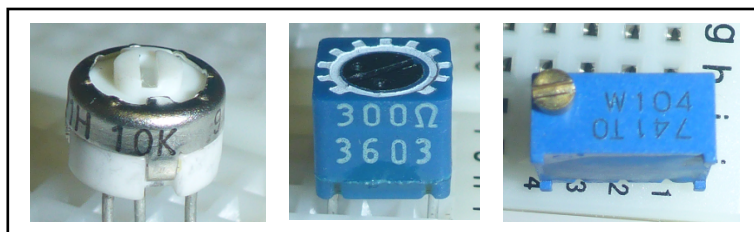


写真 6-A5 抵抗値の表記

## 6.Appendix.4 LEDの極性

LEDはリード線の長いほうがプラス側です。ブレッドボードに挿すときに切り揃えてしまうと後でわからなくなりますが、適当に挿しかまいません。極性が逆だと電流が流れず光らないだけで、LEDが壊れることはないので安心してください。

## 6.Appendix.5 測定系の回路図

電気回路図が読めるようになると、何をしているのかが理解しやすくなります。

この実験では可変抵抗器、固定抵抗器およびLEDなどが単体の部品（ディスクリートと呼びます）で組まれているので、実物と回路図（図6-A1）の突き合わせがしやすいです。ICなどの集積回路部品は内部が見えませんが、等価的な回路図を使って動作を説明することができます。

5V電源端子から可変抵抗－固定抵抗－LEDを通してGND（グラウンド、接地、全体の基準）端子に電流が流れます。

A0,A1,A2は電圧測定のための端子です。実際はA0,A1にも電流が流れ込みますが、値が非常に小さいので無視しています。

ここで大切なのはArduinoのアナログ入力は全てGNDを基準にした電位差を測定しているということです。従ってLEDの電流を検出するために必要な100Ω固定抵抗の両端電圧は、A1電位とA0電位を引き算して求めています。

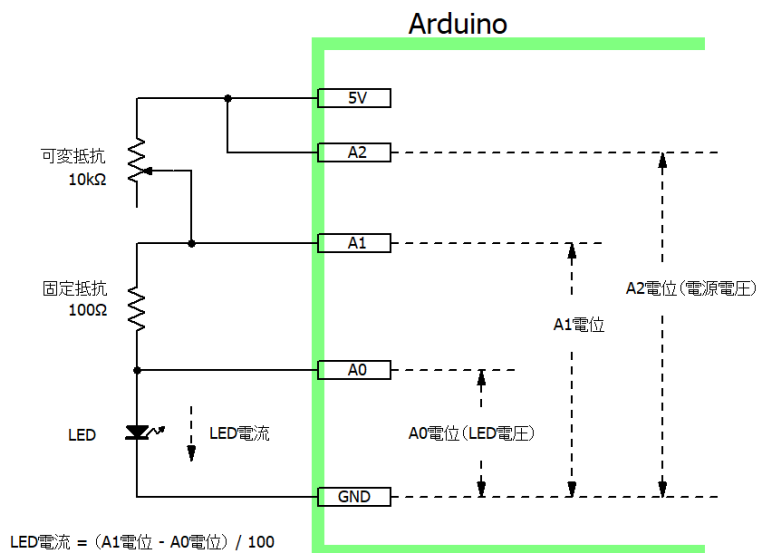
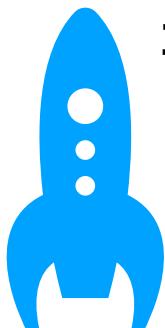


図 6-A1 実験回路図



## コラム6 LabVIEW NXGでデータ解析

取得したデータを解析することで様々なことがわかります。最近ではスマートスピーカーに「音楽をかけて」と話しかけると音楽をかけてくれますが、これは声をデータ取得し、解析したおかげです。スマートフォンをもって走ると、走った距離や消費カロリーを表示してくれるアプリがありますが、これもスマートフォンの振動を解析した結果です。「解析」というのは物事を知るうえで非常に重要です。

LabVIEW NXGでは、解析のための関数がたくさん用意されています。たいいていの解析はできると思っ  
ていただいてよいと思います。ところでこれまで解析の結果を知るためには、ある程度プログラミングを  
行ってからでなければ分らなかったのですが、LabVIEW NXGではプログラミング中にデータを解析することができ  
るようになりました。これにより、プログラミング中でも使っている関数が正しいか、こういった結果が得られるのかを確  
認できるようになったので、便利さが格段にアップしました。

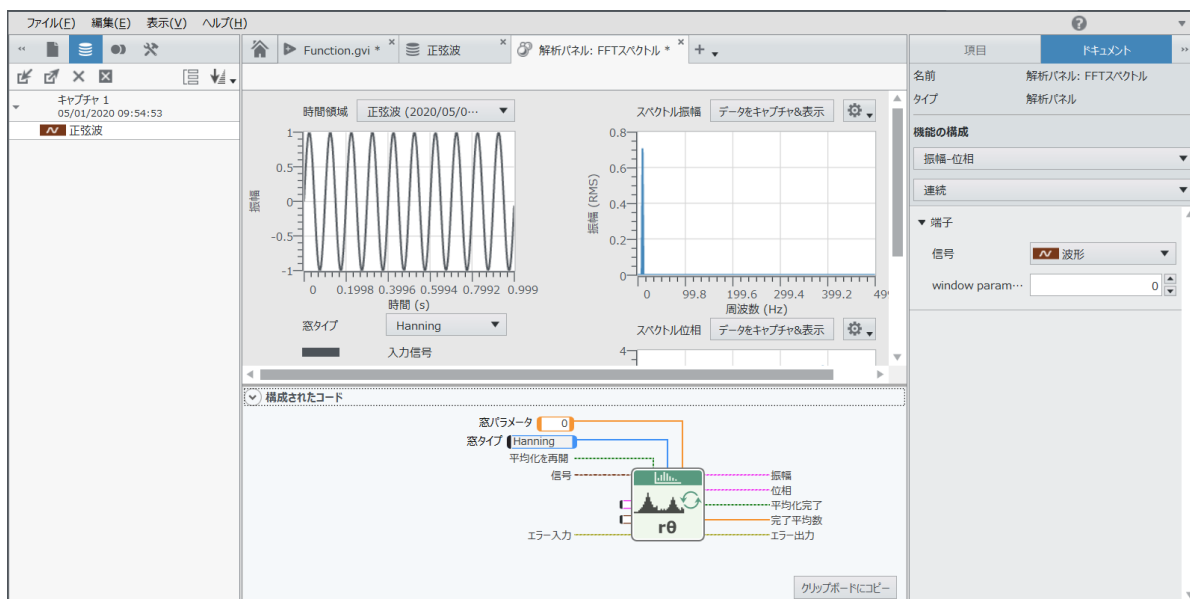


図 C6-1 LabVIEW NXGのAnalysis（解析）パネル



# 第 7 章

## 半導体センサを 使ってみる



Arduinoをセンサのインターフェースとして、LabVIEWでスマートフォンや自動車などに使用されている半導体センサを使う方法を紹介します。

**[キーワード]** 半導体センサ、心拍センサ、MAX30102、I2C、SDA、SCL、データシート、レジスタ、シリアルポート、キュー、生産者消費者デザインパターン

**[使用する部品]** Arduino UNO 1個、ブレッドボード 1個、MAX30102モジュール 1個（ヘッダピンのハンダ付け必要）、ワイヤ 4本

### 7.1 スマートフォンや自動車に使われている半導体センサ

皆さんがお使いのスマートフォンには多くの半導体センサ、半導体アクチュエータが使われています。昔から使われている原理を使っていますが半導体の微細加工技術で小型で精度が高く、さらに周辺回路との一体化によって小さくて軽いセンサやアクチュエータになりました。スマートフォンを縦に持つと縦長画面、横に持つと横長画面になったり、スマートフォンの振り方でアプリを起動したりする機能は3軸加速度センサや3軸ジャイロセンサが使われています。カラオケで手にするマイクの大きさからは想像しにくい小さなマイクも半導体の微細加工技術が使われています。通信回路に使われるRFスイッチも半導体の微細加工技術が使われ始めました。自動車にも3軸加速度センサ、3軸ジャイロセンサ、圧力センサ、流量センサ、環境(大気圧/温度/湿度)センサ、空気質センサ、車載レーダーなど多くの種類のセンサが使われていることです。

小さくて高性能な半導体センサは、昔の電気部品のように自分でハンダ付けして使うことは難しくなりましたが、周辺回路と合わせたモジュールとして購入することができます。加速度センサや環境(大気圧/温度/湿度)センサについてはArduinoでの使用例が豊富ですので簡単に使うことができると思います。この章では皮膚を通してパルス酸素濃度や心拍を測定するための機器に使われている半導体センサ**MAX30102**を使ってみます。



## 7.2 心拍数を測定するセンサ

心拍（脈拍）数を測定するMAX30102は、**写真7-1**の1円玉の右側の基板中央の素子です。発光素子と発光光量を設定する機構、受光素子、AD変換、フィルター処理、通信インターフェースを統合した高性能な半導体センサです。この素子に指先などを軽く接触させて心拍数を測定します。皮膚内部を流れている血液は心拍に対応してヘモグロビンと酸素が結合している割合（酸素飽和度）が変化します。MAX30102では酸素飽和度によって特定の光の波長の反射率が増減する現象を利用して、内蔵されたLEDから光を出して、皮膚内部で反射されて戻ってきた成分を受光素子で測定して心拍を検知しています。酸素飽和度（SpO2）も測定できるセンサですので、機会があればチャレンジしてみてください。

MAX30102モジュールは数種類販売されていますが、価格は600円台から3000円近いものまであります。**写真7-1**を参考にして購入すれば本書のピン配置で使うことができますと思います。参考までに、筆者はAmazonで送料無料で635円、翌日から数日で配送されるものを購入しました。

従来からよく使われているアナログ出力の心拍センサ（**写真7-2**）もありますが、指先や耳たぶに当てて信号を見ながら安定する場所を探り当てるのに手間取ることがありますので、本書ではMAX30102を使ってみることにしました。

なお、同梱されたピンヘッダをハンダ付けして使う必要がありますので半田ごて、ハンダが必要になります。どうしても、ハンダ付けをしたくない場合はスルーホール用テストワイヤで一時的にテストすることもできます。

## 7.3 Arduino用サンプルプログラムで動作確認

Arduino用のライブラリとサンプルプログラムを使って、MAX30102基板が正常で配線も間違っていないことを確認しましょう。

Arduino IDEの「スケッチ」メニューから「ライブラリのインクルード」さらに「ライブラリの管理…」を選択すると、「ライブラリマネージャ」ウィンドウ（**図7-1**）が表示されます。右上にある「検索をフィルタ・・・」と書かれている欄に「max3010x」と入力すると、**図7-2**のように「SparkFun MAX3010x Pulse and Proximity Sensor Library」が表示されるので「インストール」をクリックします。ライブラリマネージャを閉じてライブラリのインストールを終了します。このライブラリはMAX30100、MAX30101、MAX30102、MAX30105に対応しているとのことです。

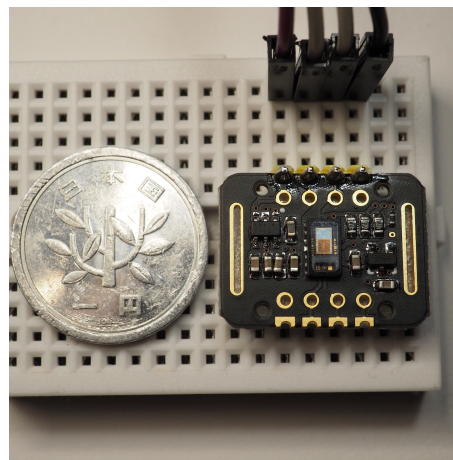


写真 7-1 MAX30102 モジュール

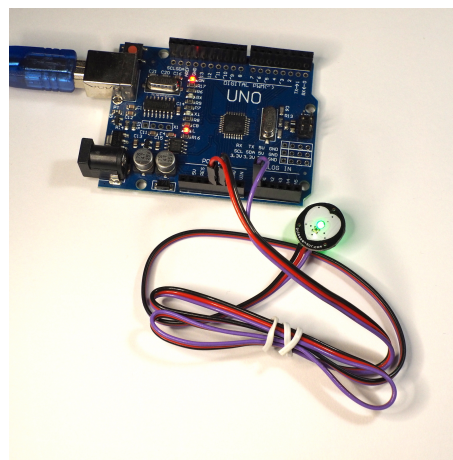


写真 7-2 アナログ出力の心拍センサ



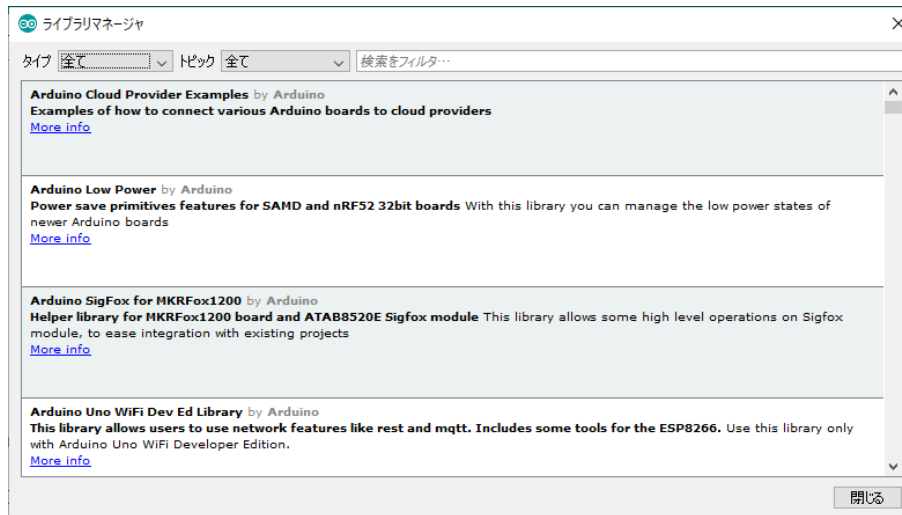


図 7-1 ライブラリマネージャ

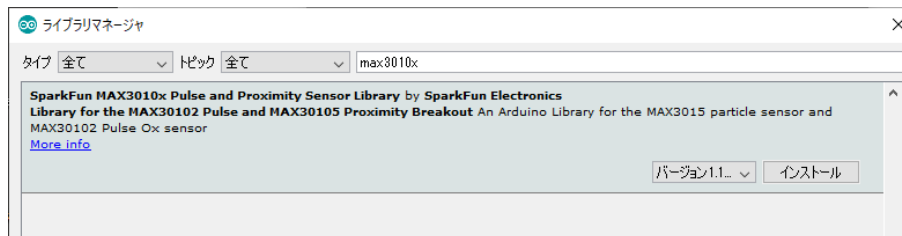


図 7-2 ライブラリマネージャMAX3010x用ライブラリ

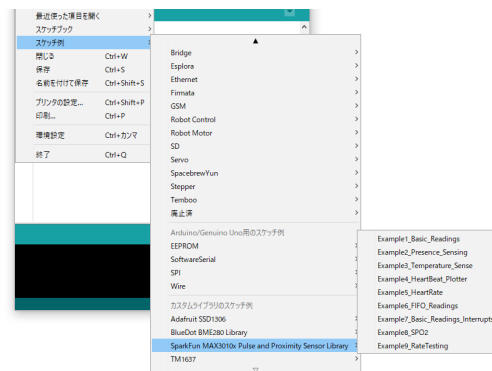


図 7-3 サンプルスケッチ

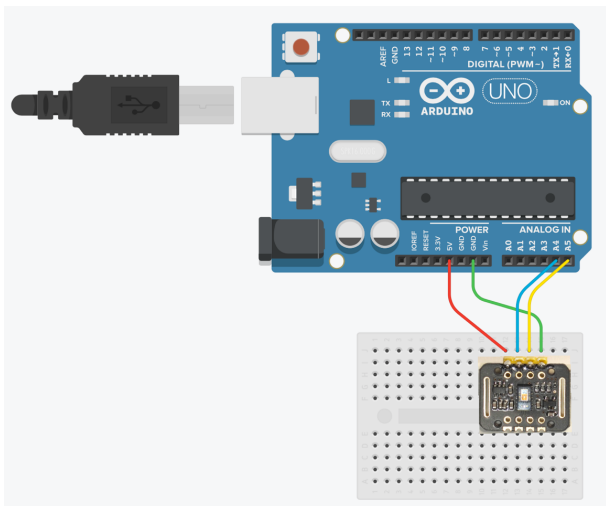


図 7-4 MAX30102の配線

「ファイルメニュー」の「スケッチ例」から「SparkFun MAX3010x Pulse and Proximity Sensor Library」の **Example4\_HeartBeat\_Plotter.ino**を開きます（図7-3）。このサンプルスケッチは心拍データをシリアルポートに出力してArduino IDEの「シリアルプロッタ」で表示するプログラムです。Arduinoへの書き込みを行ってください。ArduinoをUSBポートから外して、MAX30102をArduino UNOに接続します（表7-1）。ピン名はMAX30102の裏側に印刷されていますのでひっくり返して見ている間に右左を間違えがちです。図7-4を見ながら注意して配線してください。

ArduinoをUSBポートに取り付けて、ツールメニューからシリアルプロッタを選びます。センサに指を当てると大きなDC成分の上に心拍パルスを見ることができます。パルスを見ることができたら一安心です。

## 7.4 データシートを手に入れよう

米粒ぐらいのセンサですが高機能です。データシートを手に入れましょう。「MAX30102 datasheet」で検索するとすぐに**MAX30102.pdf**を見つけることができるでしょう。ここに基本的な使い方の情報が書かれています。英語が面倒臭いと思う人は翻訳サイトで日本語に変換しましょう。

「Typical Application Circuit」（図7-5）という回路の模式図をdatasheetから転記しました。これを見ると外部のプロセッサとはI2C通信を介してデータレジスタと読み書きすることがわかります。左端には赤色LEDと赤外LEDがあり、光センサが1個だけあります。光センサのアナログ電圧をデジタルデータに変換するADCがあります。

MAX30102	Arduino UNO
GND	GND
SCL	A5
SDA	A4
VIN	5V

表 7-1 MAX30102と  
Arduino UNOの接続ピン

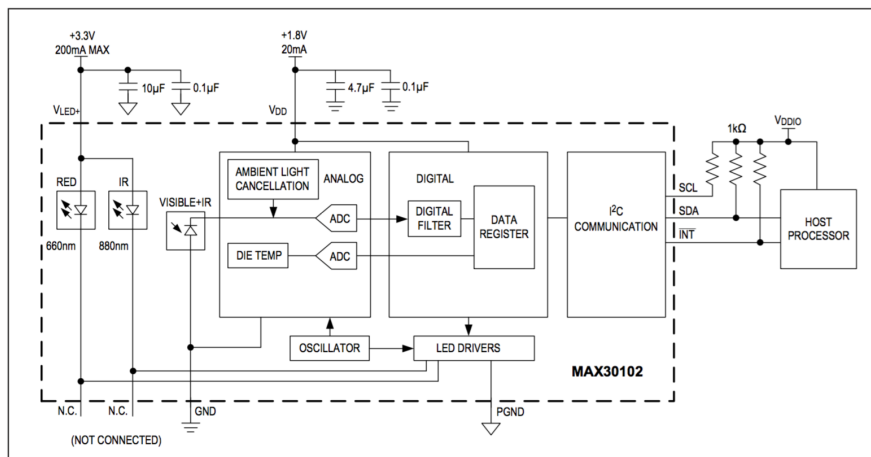


図 7-5 MAX30102模式図  
(データシートより転載)

SAMPLES PER SECOND	PULSE WIDTH (µs)			
	69	118	215	411
50	O	O	O	O
100	O	O	O	O
200	O	O	O	O
400	O	O	O	O
800	O	O	O	O
1000	O	O	O	O
1600	O	O	O	
3200	O			
Resolution (bits)	15	16	17	18

表 7-2 MAX30102設定表  
(データシートより転載)

変数	意味	Example4_HeartBeat_Plotter	my_HeartBeat10ms
ledBrightness	LEDの電流	0x1F	0x1F
sampleAverage	平均化データ数	8	8
ledMode	LEDモード	3	1
sampleRate	サンプル間隔	100	1000
pulseWidth	パルス幅	411	411
adcRange	ADC測定幅	4096	16384

表 7-3 my\_HeartBeat10ms.ino のセッティング

これで少しだけMAX30102のことがわかりました。LINXにはI2C通信の関数がありますから理詰めでこのデータシートに従って読み書きすれば動脈血酸素飽和度 (SpO2) と心拍データを得ることができます。あるいはデータシートを読み解くときのヒントとしてArduinoのライブラリをたどって理解を深めるという方法もあります。ライブラリは「ドキュメント > Arduino > libraries > SparkFun MAX3010x Pulse and Proximity Sensor Library」の中にあります。もしも見つからない時には「MAX30105」でPC内を検索すると良いでしょう。

**Example4\_HeartBeat\_Plotter.ino**はすでに動作していますから、プログラムのサンプリング速度の設定を少し変えて、意図通り変化するかどうか見てみたり、他のサンプルプログラムで動作させてみたりしながらMAX30102に親しんでいくことができます。データシートの設定表 (表7-2) に従い、サンプリング速度が速く測定レンジが広い設定を選び、

**my\_HeartBeat10ms.ino**のセッティングとしました(表7-3)。コードの不要部分を削除し、**micros()**というマイクロ秒で経過時間を返す関数を使って10msec間隔で動作するようにしました。

プログラムフォルダーから**my\_HeartBeat10ms.ino**を開いてArduinoに書き込んでください。書き込みが終了したらシリアルモニタを開いて「115200bps」で受信してみてください。6桁の数字が表示され続けることが確認できたら、Arduino IDEを終了してください。

## 7.5 LabVIEWシリアル受信プログラムの作成

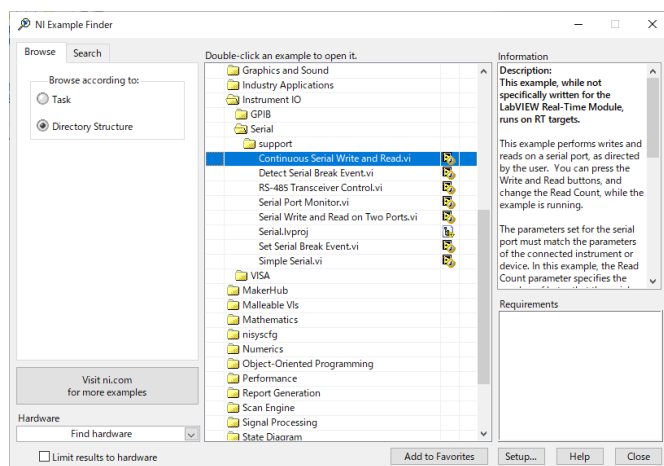


図 7-6 NI エグザンプルファインダ

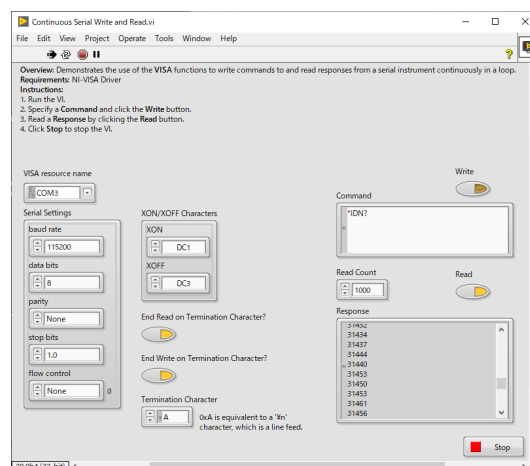


図 7-7 Continuous Serial Write and Read.vi

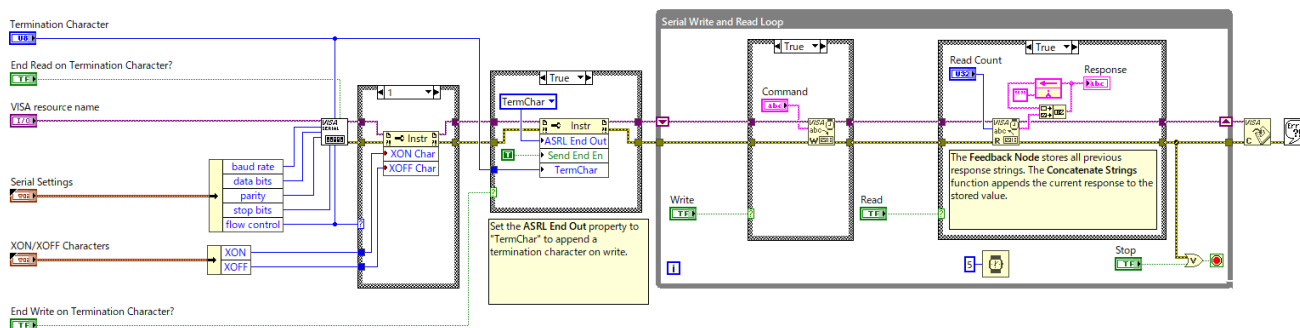


図 7-8 Continuous Serial Write and Read.viのブロックダイアグラム

LabVIEWには多くのサンプルプログラムがありますが、その中にArduino IDEのシリアルモニタのような機能を持ったものがあります。NI エグザンプルファインダでDirectory Structureを選択してください。目的のVIは、「Instrument IO > Serial」にある**Continuous Serial Write and Read.vi**です（図7-6）。ダブルクリックで開いて、プログラムを作成するフォルダに保存してください。Arduinoが使用しているポート番号とボーレート「115200」に変更して実行ボタンを押します。Readボタンを押すとシリアルモニタのように文字列が表示されるはずです（図7-7）。このサンプルVIは機能が多いのでブロックダイアグラムは複雑そうに見えます（図7-8）。今回使わない機能を削除して**Continuous**

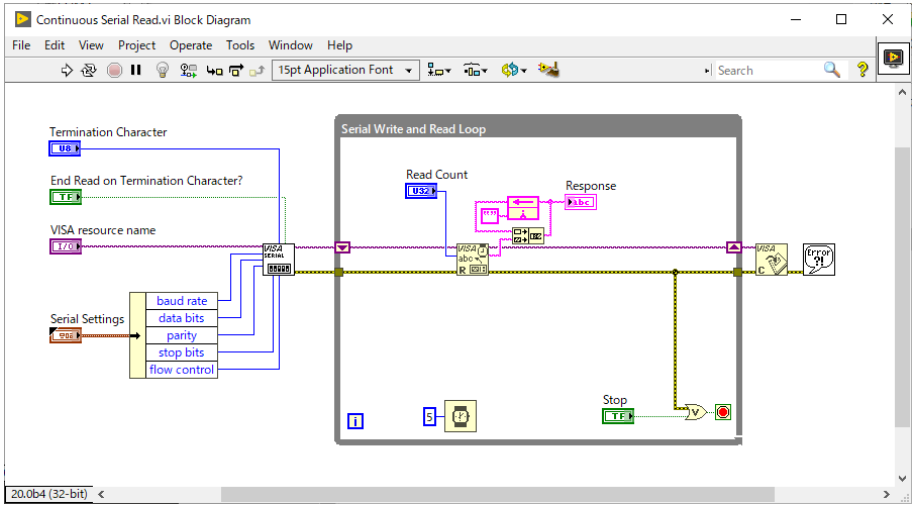


図 7-9 Continuous Serial Read.vi

項目	値	補足
送信文字列長	8バイト	送信文字 改行コード (CR+LF) 6文字
1文字あたりの 通信ビット数	10ビット	データビット 8ビット スタートビット 1ビット ストップビット 1ビット
総通信ビット	80ビット	送信文字列長 X 1文字あたりの 通信ビット数
通信速度 (bps)	115200 bps	bps:ビット/秒
通信時間 (sec)	0.00069sec	総通信ビット / 通信速度

表 7-4 シリアル通信に要する時間の見積もり

**Serial Read.vi**として保存します（図7-9）。この数字の文字列を数値に変換すれば加減乗除やグラフに表示することができます。

Arduinoから10msecごとに最大6桁の数値と2文字分の改行コード（CRとLF）が送られてきますので、シリアル通信に必要な時間を表7-4で見積もってみるとおよそ0.7msecでした。シリアル通信でよく使われる「9600bps」という通信速度の場合を計算してみてください。その場合は少ししか余裕がなくなってしまうです。データを受け取りながらデータ処理やデータ保存していると、場合によっては時々データを受け取りそこなうこともできます。そのような状況の時には生産者消費者デザインパターンを使います。データを受け取る仕事とデータ処理する仕事を分けることで、お互い独立して仕事ができるような仕組みです。

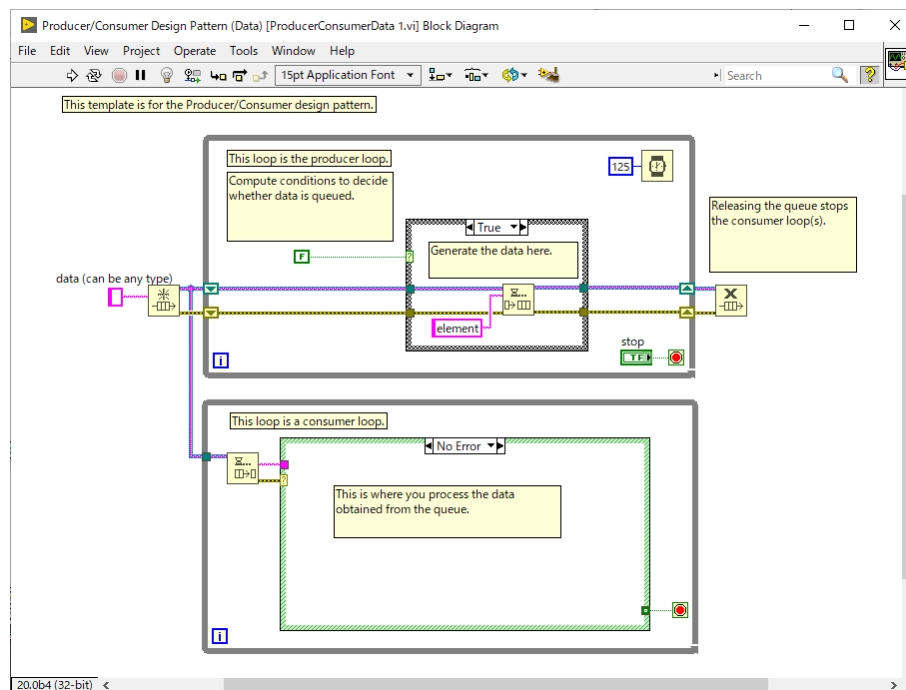


図 7-10 Producer/Consumer Design Pattern (Data).vi

今回のMAX30102を使った心拍測定プログラムでは余裕がありますが、生産者消費者デザインパターンを使ってみましょう。ファイルメニューからNewを選んで「From Template > Frameworks > Design Patterns」の中の**Producer/Consumer Design Pattern (Data).vi**を開きます。ブロックダイアグラム（図7-10）には2個のWhileループがあり、上のループで受信した文字列をキューで下のループに伝えています。キューは人気店の前の行列みたいなもので、下の処理ループが時々滞っても行列が少し長くなるだけで、取りこぼしなく処理することができます。処理ムラに対応するだけなので、平均的に処理に時間がかかりすぎる場合はいずれ破綻しますが、グラフ表示などのUIを操作した時に処理が遅くなってしまうような一時的な遅れには対応できます。

**max30102ChartDisplay.vi**という名前で作成フォルダに保存します。**Continuous Serial Read.vi**から**VISA Configure Serial Port**、**VISA Read**、**VISA Close**を接続されている制御器、表示器ごと上のループにコピー＆ペーストしてください。図7-11のようになります。このままでもVIは動作しますが、下のループとは無関係に動作するだけです。3か所変更します。

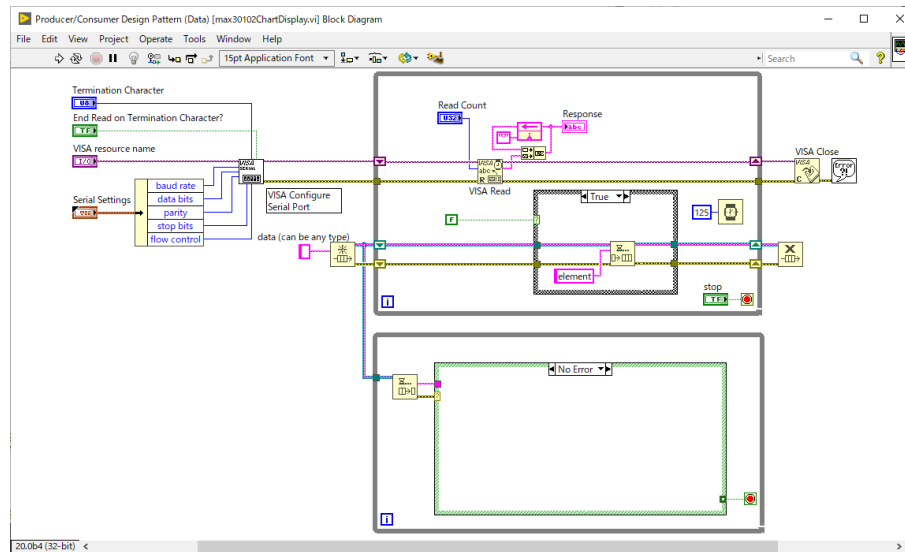


図 7-11 Continuous Serial Read.viからVISA関数をコピー

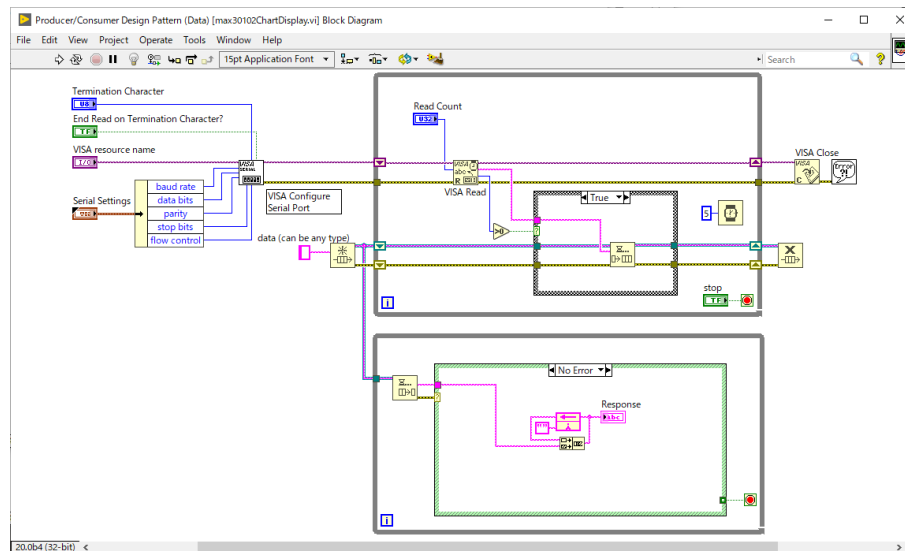


図 7-12 受信文字列をキューで渡す

(1) **VISA Read**から出力された文字列を受け取っている部分を下のループの「No Error」ケースストラクチャに移動してください。**Dequeue Element**から出力されている文字列に配線してください。

(2) VISA Readの接続されていない出力「return count」は読み取った文字数を出力するので「0」より大きい時に**Queue Element**が入っているケースが選択されるようにケース端子に**Greater Than 0?**を接続します。

(3) **Wait(ms)**に接続された定数を「5」に変更します。

図7-12のようになります。VIを実行させてみてください。生産者消費者デザインパターンで動作しています。ここまでは数字の文字列で送られてきたものをそのまま文字列で表示しているだけです。フロントパネルに「Graph」パレットから「Waveform Chart」をドラッグドロップしてください。

ブロックダイアグラムで数字の文字列を数値に変換します。

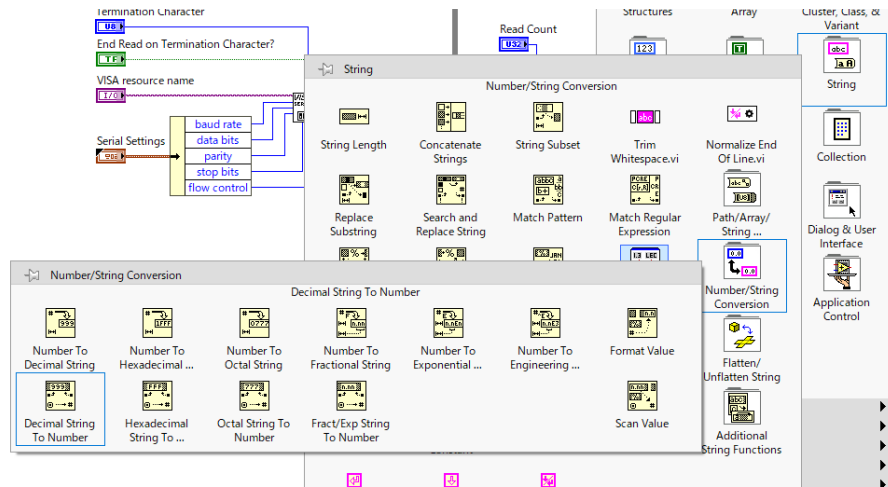


図 7-13 Decimal String To Number

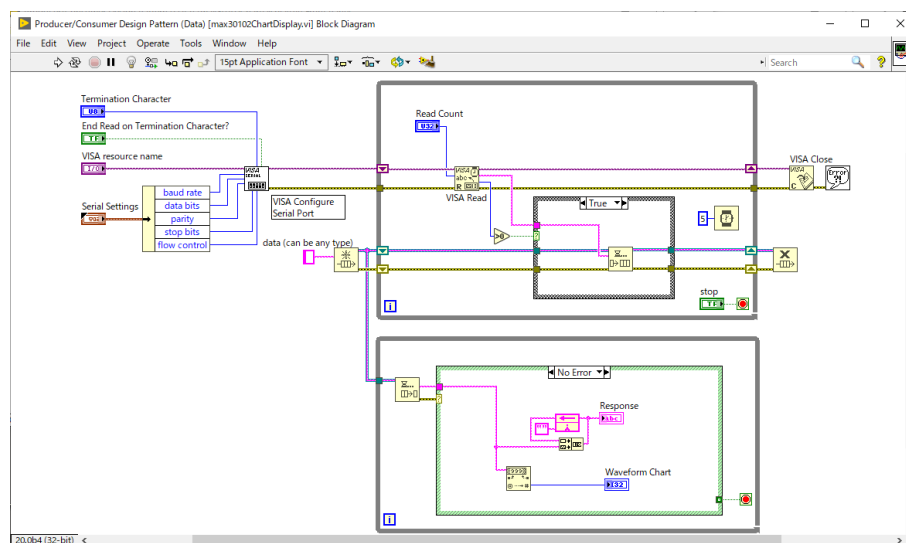


図 7-14 文字列を数値に変換してWaveform Graphに表示する



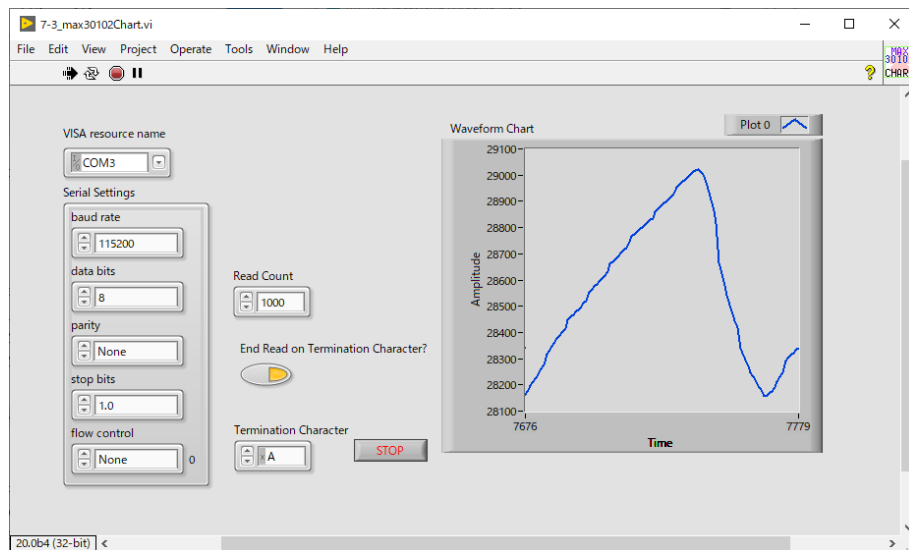


図 7-15 VIを実行して血流パルスを表示させる

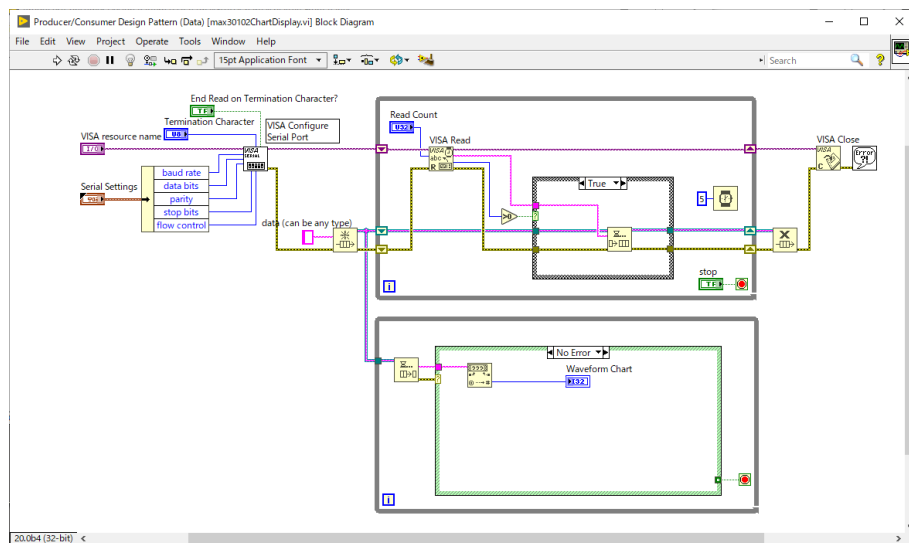


図 7-16 Response表示器への表示を削除する

String Conversion」にある**Decimal String To Number** (図7-13) をドラッグドロップして図7-14のように配線します。実行すると図7-15のように血流パルスが表示されます。

このプログラムはしばらく実行していると動作がギクシャクしてくると思います。「Response」という文字列表示器に文字がたまりすぎてしまうためです。データは「Waveform Chart」で見ることができるので「Response」は削除してしまいます。エラークラスターの配線を1系統にまとめて、図7-16で一区切りです。

ここで、「STOP」ボタンを押した時にどんな順番で停止するか、「実行のハイライト」を押して停止のメカニズムを観察してください。

## 7.6 心拍数測定プログラムの作成

**max30102ChartDisplay.vi**の下ループに心拍数を求める処理を追加しますので、**MAX30102\_Plot.vi**という名前で保存してください。心拍数は1分間のパルス数ですが、10秒間のデータを配列にして周波数解析により基本周波数を求めて1分間のパルス数を推定したいと思います。周波数解析をするためにはデータがサンプリングされた間隔が必要になります。Arduinoからは10msec間隔でデータが送られてきますのでサンプリング間隔「dt」は0.01secとなります。10秒分のデータ数は1000個ですので、要素数1000の配列を用意します。図7-17が要素1000個の配列を作ってシフトレジスタに接続した状態です。表示されていませんが、「Error」サブダイアグラムにもワイヤーを配線しています。

波形データと同じ考え方で最新のデータが常に配列の末尾になるようなデータにしたいと思います。**Rotate 1D**

**Array.vi**は配列が環（リング）のように先頭と最後尾が繋がったものと考えて要素を順番に前に移動する関数です。何個

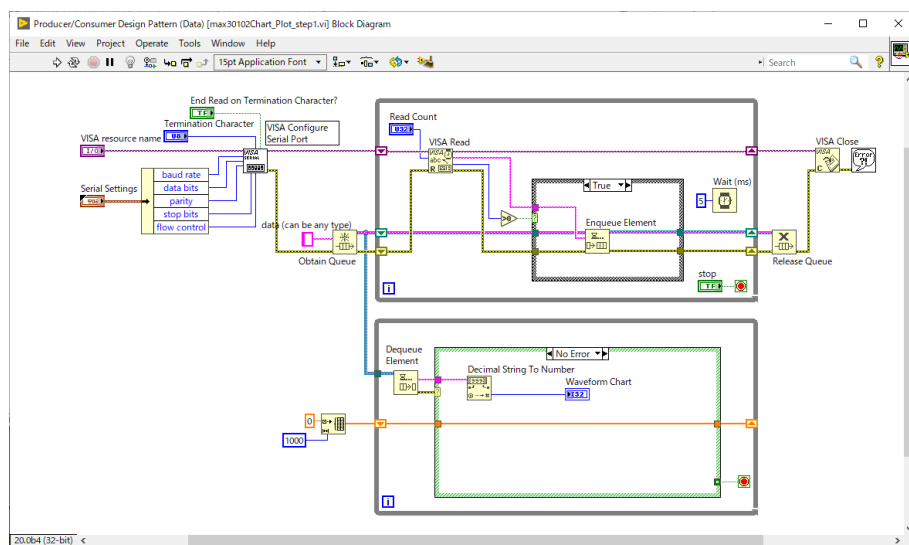
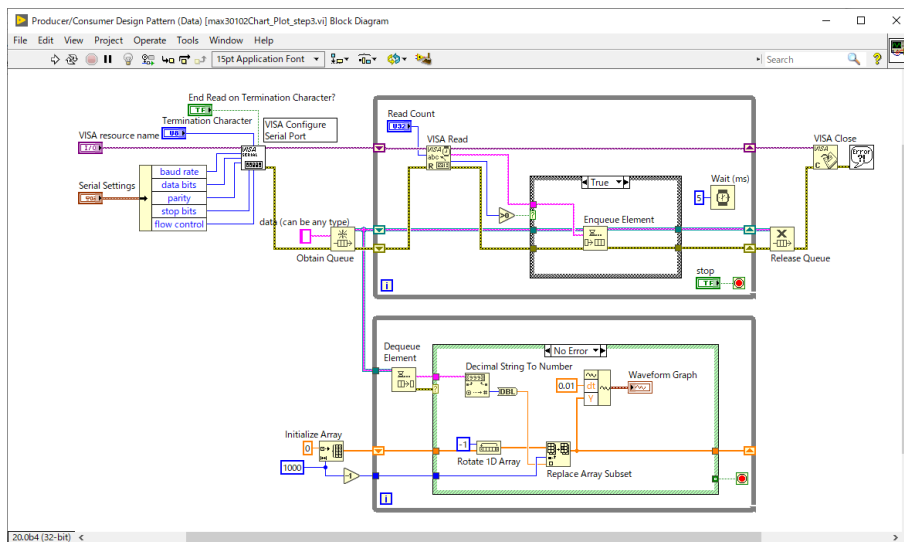
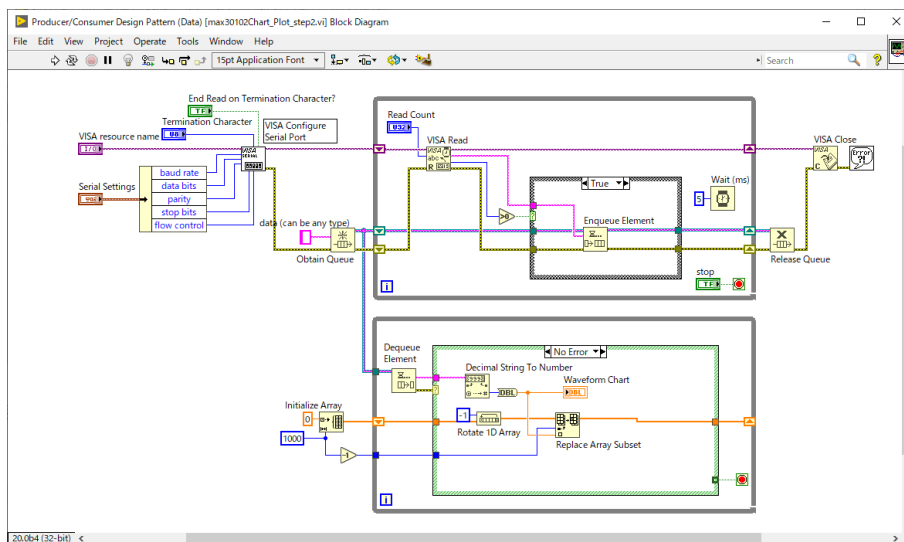


図 7-17 下のループに配列とシフトレジスタを作成





移動するか指定することができて、「1」を入力すると最後尾の指標「999」の要素が指標「0」に移動し、指標「0」の要素は指標「1」に移動します。ここでは「-1」を入力して指標「0」が最後尾である指標「999」に移動するようにします。指標「999」の要素は指標「998」に移動して、・・・、指標「1」の要素が指標「0」に収まります。この関数のアイコンはアイコンを見れば機能がわかる良い例です。さて、配列の要素がぐるりと回った後で、指標「999」を新しい値に入れ替えます。このような使い方は**リングバッファ**と呼ばれています（図7-18）。

図7-19ではサンプリング間隔とデータ配列から波形データを作って「Waveform Graph」に接続しました。

VIを実行すると図7-20のように表示されます。

波形データを**Extract Single Tone Information.vi**に入力すると周波数解析を行って一番顕著な周波数を出力します。60倍することで1分間の心拍数を得ることができます（図7-21）。図7-22が表示画面です。

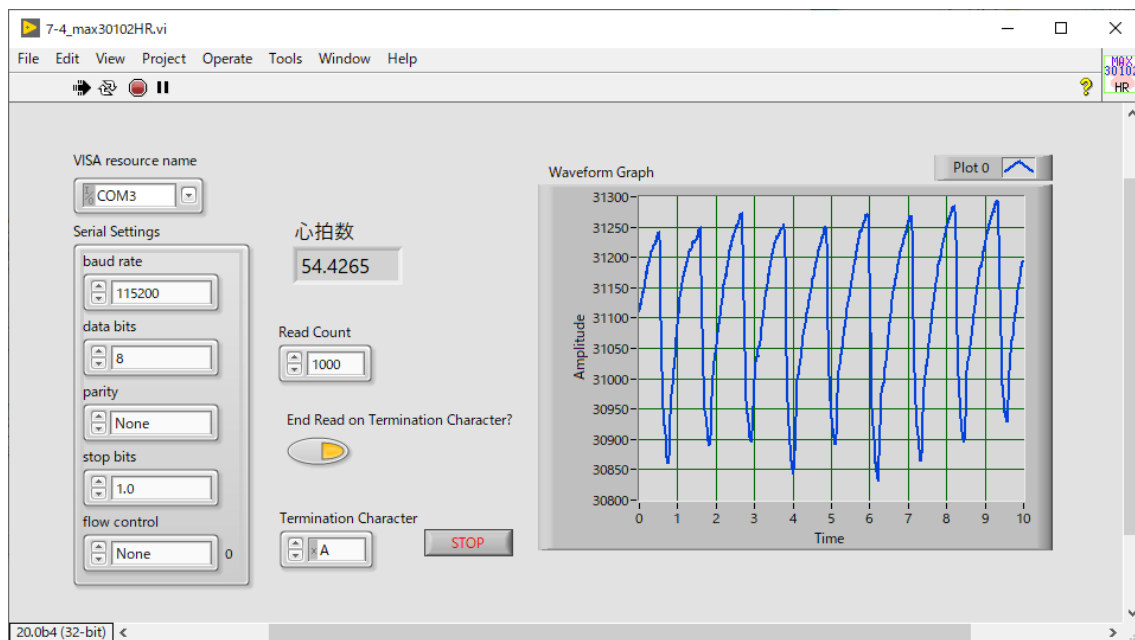
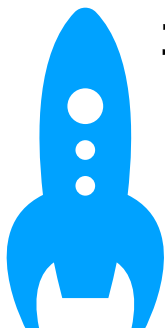


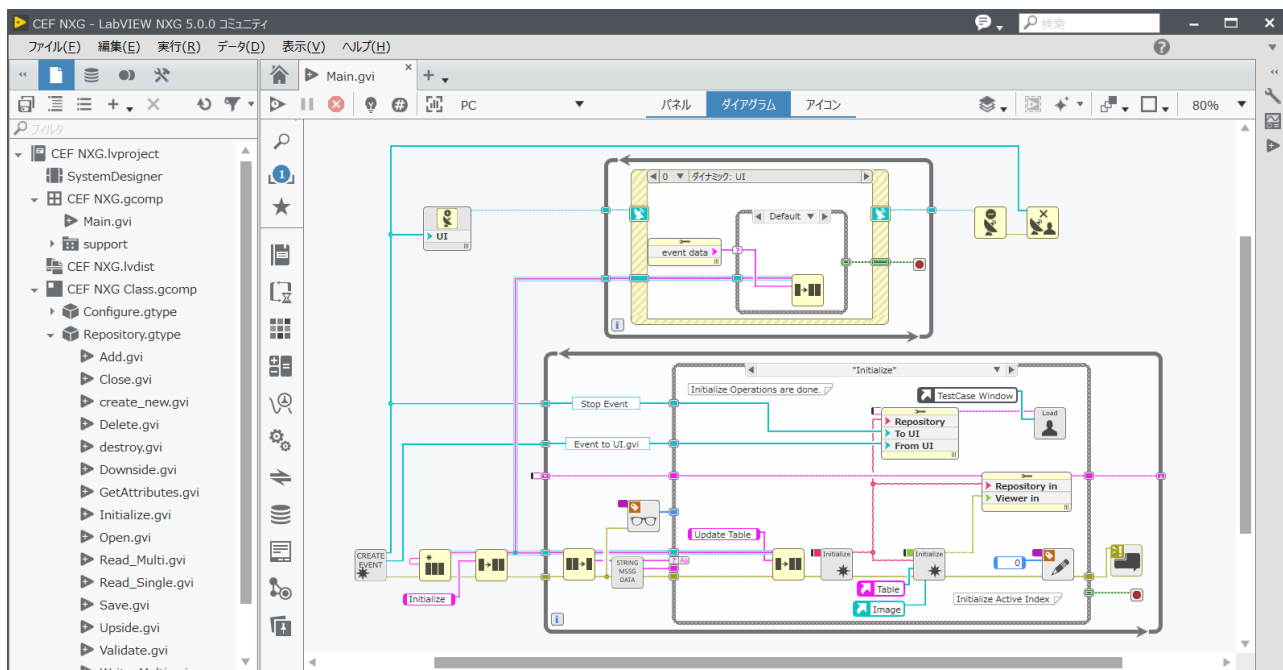
図 7-22 心拍数の表示画面



## コラム7 LabVIEW NXGの目指すもの

技術変革はとても早いスピードで進んでいて、昨日までの最新技術が今日は昔のモノなんてこともあったりします。技術の複雑化の一方で、より早いスピードで新製品・新技術を世に出すことが求められています。LabVIEW NXGは製品開発における必要な計測や制御を、より素早く実現できるように作られています。これまでのコラムでもプログラミングの前にまず集録、まず解析ができる点をご紹介しましたね。

LabVIEW NXGはまだ発展途上ではありますが、最終的にはLabVIEWが持つすべての機能をLabVIEW NXGが持ち、かつLabVIEWにはない様々な機能が実装されると期待されています。まずはLabVIEWに慣れて頂き、今後LabVIEW NXGがメジャーとなってきたら、LabVIEW NXGに乗り換えることも是非ご検討ください。



☒ C 7-1 LabVIEW NXG

# サンプルVIのリスト

使用する章	プログラム名	簡単な説明
3章	3-1 LED Simulator.vi	LEDの特性（順方向電圧、最大電流）を設定して電源電圧や電流制限抵抗によるLED電流変化をシミュレートするプログラムです。
3章	3-2 my_LED_step1.vi	電源電圧、順方向電圧、電流制限抵抗からLEDに流れる電流を計算するプログラムです。
3章	3-3 my_LED_step2.vi	“順方向電圧”制御器の入力を負の数が入力できないように制限したプログラムです。
3章	3-4 my_LED_step3.vi	LEDを流れる電流が、LEDの最大電流値よりも大きければ“LED破損?”表示器を点灯するプログラムです。
3章	3-5 my_LED_step4.vi	Whileループを使って連続的に動作するプログラムです。
4章	4-1 SoundVIEW.vi	マイク、PC音源からの音声信号を録音し、再生時に周波数解析を行い、スペクトログラムを表示するプログラムです。
4章	4-2 AvailableDevice.vi	LabVIEWが認識している音源を表示するプログラムです。
4章	4-3 Finite Sound Input.vi	音声信号を取り込むプログラム例としてNIが提供しているサンプルVIです。4章で改造して使用します。
4章	4-4 parrot.vi	取り込んだ音声信号を一定時間後に再生するプログラムです。
4章	4-5 reverse.vi	取り込んだ音声信号を時間を反転して再生するプログラムです。
4章	4-6 ForLoopSum.vi	1からNまでの合計を求めるプログラムです。
4章	4-7 PrimeNumber.vi	指定する正の整数よりも小さい素数を全て求めるプログラムです。
4章	4-8 2D_array_Display.vi	2次元配列の多様な表示方法を紹介するプログラムです。
4章	4-9 PiPoPa.vi	電話のトーンダイヤルの音を発生させるプログラムです。

使用する章	プログラム名	簡単な説明
4章	4-10 easyRecorder.vi	ステートマシンを使った録音再生ができるプログラムです。
5章	5-1_PushCounter.vi	ArduinoとLINXを使った、スイッチを押すと1ずつ増えて、10を超えると0に戻るカウンタプログラムです。
5章	5-2_FanControl.vi	カウンタでファンの回転を制御するプログラムです。
5章	5-3_FanControlWith LongPushStop.vi	スイッチを長押しするとファンが停止するプログラムです。
5章	5-4_FanControlWith LongPushImmediate Stop.vi	スイッチを長押しするとファンがすぐに停止するプログラムです。改良が必要です。
5章	5-5_Push Counter (LatchWhenReleased).vi	5-2と同じ動作をするVIを別のアルゴリズムで作ったカウンタプログラムです。
5章	5-6_Fan Control with Long Push Stop (LatchWhenReleased).vi	5-4と同じ動作をするVIを別のアルゴリズムで作った、スイッチを長押しするとファンがすぐに停止するプログラムです。
6章	6-1_LINX - Analog Read N Channels.vi	LINXのアナログ入力のプログラム例としてMakerHubが提供しているサンプルVIです。6章で改造して使用します。
6章	6-2_LED VI Curve2.vi	LEDの電圧電流特性を測定して、回帰分析を行うプログラムです。
7章	7-1_Continuous Serial Write and Read.vi	シリアル通信のプログラム例としてNIが提供しているサンプルVIです。7章で改造して使用します。
7章	7-2_Continuous Serial Read.vi	シリアル通信のサンプルVIを受信専用に変化したプログラムです。
7章	7-3_max30102Chart.vi	Arduinoからの心拍信号を受信して表示するプログラムです。生産者・消費者デザインパターンを使っています。
7章	7-4_max30102HR.vi	Arduinoからの心拍信号を受信して心拍数を表示するプログラムです。生産者・消費者デザインパターンを使っています。
7章	Example4_HeartBeat_Plotter	ArduinoでMAX30102心拍センサを使うためのライブラリで、SparkFunが提供しているArduino用サンプルプログラムです。
7章	my_HeartBeat10ms	シリアル通信を使って心拍データを10msec間隔で送信するArduinoプログラムです。



# あとがき

LabVIEWはいわゆる「プログラミング言語ランキング」には入ってきません。それは調査対象がプログラマだからです。LabVIEWは科学者や技術者などプログラミングが専門でない人向けに発明されました。仕事の効率を上げるために自動で作業してくれるプログラムや装置を作るときにとっても便利です。自分の趣味のなかでもちょっとパソコンの力を借りて便利にできたら楽しいと思いませんか？ でもちょっとのためにプログラマ向けの言語を勉強するのは大変ですね。そんなときLabVIEWが助けてくれます。

「自動で作業してくれる装置」はいろいろな業界で必要とされ、それを仕事にしている人もたくさんいます。LabVIEWはそのようなプロの間で最も使われています。

日本LabVIEWユーザー会はLabVIEWが大好きな人たちの集まりで、メンバーは日々LabVIEWを使って仕事や趣味を"楽しんで"います。そしてこの素晴らしいツールをもっとたくさんの人に使ってもらいたいと願っています。しかしプロ向けで値段も安くはないためか、広く普及しているとは言えないのが残念で仕方ありません。

そんなある日「LabVIEWプロフェッショナル版が無料で使えるようになるらしい」という話が飛び込んできました。これでLabVIEW仲間が爆発的に増えるかもしれない。ベータテストが始まり期待が高まったころ「コミュニティ版で初めてLabVIEWを手にする人向けのドキュメント」を作ろうということになりました。もちろん無料。中高生向けの資料を目指せばわかりやすい内容になるのでは？ コンピュータと現実世界が物理的につながる「フィジカルコンピューティング」が最もプログラミングの楽しさを体感できるはず。正式リリースが予定される5月に合わせて発表。そんな方針でプロジェクトがスタートしたのが2020年1月末のことでした。

有志によってアウトラインが提示され、パート毎にテキストが書き起こされサンプルVIが作られます。別のメンバーがテキストに沿って実験を行い再現性をチェックします。文章校正、追加コラムの執筆、図の作成、レイアウトなどを経てできたのが本書です。少しでもLabVIEWへの愛を感じ取っていただけたら幸いです。

あなたがLabVIEWを気に入ってくれることを願っています。そして本書にないVIだったりあなたの得意分野の視点やアイデアを具体化した記事ができたら、今度はそれを私たちや他のユーザーに教えてください。一緒に楽しみましょう。

日本LabVIEWユーザー会会長 渡島浩健

## **制作：日本LabVIEWユーザー会有志**

著者：朽木 佑輔（はじめに、第1章から第3章）

著者：大橋 康司（第4章から第7章）

著者：渡島 浩健（第6章Appendix、あとがき）

表紙デザイン・写真：大橋 康司

企画：朽木 佑輔

運営：朽木 佑輔

構成：大橋 康司

編集：大橋 康司

校正：渡島 浩健、江村 秀俊

プログラム作成：大橋 康司

プログラム改良：渡島 浩健

（VI Properties > Documentationに記載）

プログラム動作確認：渡島 浩健、江村 秀俊

運営協力：鴨志田 敦史

日本ナショナルインスツルメンツ株式会社

Sales本部 第二営業部 アカウントマネージャー 先端科学研究領域担当

発行： 2020年4月28日（バージョン0.9.0）

2020年 5月 7日（バージョン1.0.0）

2020年 6月 24日（バージョン1.1.0）

2021年 10月 20日（バージョン2.0.0）