I/0ボード制御プログラムをJavaScriptで試作するためのライブラリ

平成 29 年度

学士学位論文

I/O ボード制御プログラムを JavaScript で試作するためのライブラリ

A Library For Prototype I/O Board Control Program Using JavaScript

1180298　　大石 孝文

指導教員　　鵜川 始陽

2018 年 2 月 28 日

高知工科大学 情報学群

要 旨

I/O ボード制御プログラムを JavaScript で試作するためのライブラリ

大石 孝文

IoT の発展により組込みシステム開発の需要が増加している．組込みシステムを開発するプログラミング言語としてよく使われているものが，アセンブリ言語や C 言語などの低級言語である．しかし，ハードウェアとメモリを管理する必要があるため，低級言語での開発は難しく，誤りを含みやすい．

高知工科大学情報学群の 2 年次に開講されている講義でも，RaspberryPi に接続された講義用 I/O ボードをアセンブリ言語で制御している．この講義では，RaspberryPi をアセンブリ言語でプログラムし，講義用 I/O ボードに付属している装置である LED，ディスプレイ，スイッチ，スピーカーを制御する．アセンブリ言語によるプログラミングでは，単純な I/O 制御を記述するだけでも，デバイスとメモリの制御レジスタを直接処理する必要がある．そのため，プログラムを書くのに時間がかかるうえ，誤りも起こりやすい．このような理由から，短時間でプログラムを試作することは難しい．JavaScript のような高級言語で開発すれば，簡単にプログラムが書けるようになり，この問題は解決できる．

そこで本研究では，JavaScript で講義用 I/O ボードを制御できるようにすることで，この問題を解決した．まず，RaspberryPi に JavaScript 仮想機械を導入した．JavaScript 仮想機械には，本研究室で開発している組込み用 JavaScript 仮想機械である eJSVM を使用した．さらに，講義用 I/O ボードの制御を容易にするために，eJSVM 用の講義用 I/O ボードを制御するライブラリを作成した．本ライブラリは講義用 I/O ボードの各装置を操作する基本的な関数を備えている．これらの関数は，組込みオブジェクト `Raspi` の組込みメソッ

ドとして実装した．

本ライブラリの有用性を確認するために二つの実験を行った．まず，本ライブラリを使用し，講義用 I/O ボードの全ての装置を制御する事が可能なのか確認した．講義の全ての演習課題を，本ライブラリを使った JavaScript で作成した．その結果，本ライブラリは講義用 I/O ボードの全ての装置を制御できることが確認できた．次に，講義用 I/O ボードの制御を JavaScript と本ライブラリを使用した場合，アセンブリ言語を使用するよりも簡単になったかどうかを調べた．講義用 I/O ボードを使う講義を履修した 6 人の学生を被験者とし，JavaScript と本ライブラリにより講義用 I/O ボードを制御するプログラムと，アセンブリ言語により講義用 I/O ボードを制御するプログラムをそれぞれ作成させた．その結果，本ライブラリを使用した場合の全ての被験者においてプログラムの作成時間が短縮され，開発中にプログラムを修正する回数も少なかった．さらに，開発されたプログラムも JavaScript と本ライブラリを使用した方が短くなった．

キーワード　組込みシステム, ライブラリ，JavaScript

Abstract

A Library For Prototype I/O Board Control Program Using JavaScript

Takafumi OISHI

As improvement IoT technology, demand for embedded system development is increasing. Commonly used programming language for development embededd systems are low-level languages such as assembly language and C language. However, development with a low-level language is difficult and error-prone because we have to be aware of hardware and memory management. In the lecture for second year students of the school of Information of Kochi university of technology, students develop programs controlling Raspberry Pi connected to the dedicated I/O board in assembly language. In this lecture, students controls the LED, the display, the switches, and the speaker implemented on the I/O board.

Programming with assembly language requires to handle control registers of devices and memory directly even when writing a simple I/O controling. Therefore, writing in assembly language is time-consuming and error-prone. For this reaseon, it is difficult to prototype a program in a short time. If develop it in a high-level language such as JavaScript, it will be easy to write programs, and this problem can be solved.

In this research, we addressed this problem by enabling users to develop programs controlling the I/O board in JavaScript. First, we introduced a JavaScript virtual machine to Raspberry Pi. We used eJSVM, which we are developing in our laboratory, for the JavaScript VM. Second, in order control the I/O board easily, we developed a

library for eJSVM to control the I/O board. This library provides fundamental functions to control each device on the I/O board. These functions are implemented as built-in methods of the built-in object, `Raspi`. We conducted two kinds of experimentation to evaluate effectiveness of our library. First, we confirmed that every device can be controlled by using our library. For that, developed programs in JavaScript with our library for all exercises of the lecture. As a result, we confirmed that our library covers all devices of the I/O board. Second, we investigated whether development of the I/O board in JavaScript with our library is easier than that in assembly language. We have six students develop a program controlling the I/O board both in JavaScript with our library and in assembly language. As a result, all students developed in shorter time and with fewer error corrections in the development when they use JavaScript. In addition, the programs developed in JavaScript were shorter than those in assembly language.

key words embedded system, library, JavaScript

目次

目次

図目次

表目次

第 1 章

はじめに

IoT の発展により組込みシステム開発の需要が高まっている．組込みシステムの開発にしてよく使われているプログラミング言語は，アセンブリ言語や C 言語などの低級言語である．しかし，低級言語での開発は難しく，誤りを含みやすい．

高知工科大学情報学群の 2 年次に開講されている講義では，RaspberryPi に接続した講義用 I/O ボード (以下，KUT-I/O ボード) をアセンブリ言語で制御する演習を行う．この講義では，RaspberryPi に接続した KUT-I/O ボードをアセンブリ言語でプログラムし，KUT-I/O ボードに付属している装置である LED，ディスプレイ，スイッチ，スピーカーを制御する．本来，I/O 制御を行う場合は OS に組み込まれているデバイスドライバによって操作するが，講義では OS にたよらずに CPU 上でプログラムを動作させ，装置の制御回路に直接命令を出している．そのため，ユーザーがプログラム中に制御を行う命令を書く必要がある．図 1.1 は，KUT-I/O ボードの機能の中でも比較的簡単に制御できる LED の点灯を行うプログラムをアセンブリ言語で書いたものである．

このプログラムでは 6 行目や 7 行目で，メモリ番地に値を書き込んで，LED 制御レジスタを操作している．これだけのプログラムでも，命令の抽象度が低く，制御レジスタとメモリ番地との対応を意識する必要があり，難しい．そのため，プログラムを書くのに時間がかかるうえ，誤りも起こりやすい．このような理由から，短時間でプログラムを試作することは難しい．

本研究では，KUT-I/O ボードを制御するプログラムを JavaScript で書けるようにすることを目標とする．JavaScript のような高級言語で制御するプログラムを開発できれば，制御レジスタとメモリ番地との対応の制御を意識することなく抽象的にプログラムを，適当に

```
 1     .section .init
 2     .global _start
 3 _start:
 4     ldr     r0, =0x3f200000
 5
 6     mov     r1, #(1 << 0)
 7     str     r1, [r0, #0x04]
 8
 9     mov     r1, #(1 << 10)
10     str     r1, [r0, #0x1c]
11
12 loop: b       loop
```

図 1.1　アセンブリ言語で書いた LED 制御プログラム

```
1       Raspi.light();   // LED の点灯
```

図 1.2　関数を使って LED を点灯させる JavaScript プログラム

設計されたライブラリを用いて記述できる．本研究は，以下の 2 つの方針で行った．

- JavaScript 処理系には eJSVM を利用する
- KUT-I/O ボード専用のライブラリを開発する

JavaScript 処理系には本研究室で開発している eJSVM を用いる．eJSVM は組込み用に特化した JavaScript 仮想機械である．仮想機械のプログラムサイズが小さいため RaspberryPi のような計算資源の少ない組込み機器に適している．また，KUT-I/O ボードの全ての装置を制御するためのライブラリを作成する．ライブラリは eJSVM の組込み関数として作成する．例えば，図 1.1 に示す LED を点灯させるプログラムは，本研究で作成するライブラリを使って図 1.2 のように記述できるようにする．図 1.2 の `Raspi` は組込みオブジェクトであ

り，`light()` は組込みオブジェクトのメソッドである．このように `Raspi` オブジェクトメソッドで KUT-I/O ボードの各装置を制御する．

本論文の以降の章は次のようになっている．2 章では本研究の関連研究として JavaScript エンジンである JerryScript と RaspberryPi を制御する Python ライブラリについて説明する．3 章では本研究で開発している組込み用 JavaScript 仮想機械である eJSVM について説明する．4 章と 5 章では本研究で作成したライブラリの設計と実装を説明する．6 章では作成したライブラリを使った評価実験について説明する．

第 2 章

関連研究

2.1 JerryScript

組込みシステムを制御するプログラムは，JavaScript を用いると記述しやすいと言われる [1]．普通，組込みシステムで I/O を制御する場合，低級言語でハードウェアを直接制御するプログラムを記述する．しかし，JavaScript で開発できるようになれば，抽象的にプログラムを書けるようになり，開発効率の向上やバグの混入を減らすことができる．そこで，組込みシステムの開発に利用できる JavaScript 処理系が開発されている．

JerryScript は，JavaScript で組込みシステムを開発する事を目的に Samsung によって開発されている組込み機器向けの JavaScript エンジンである [3]．一般に，組込み機器は CPU 性能やメモリなどの計算資源が少ないなどの厳しい制限がある．JerryScript は 64KB 以下の RAM で動作させることができ，コード全体で 200KB の ROM に収まる JavaScript エンジンとなっている．また，メモリ消費量を大幅に最適化することが可能であるため，計算資源の少ない組込み機器に適している．

JerryScript の API の中には，RaspberryPi の GPIO を制御するためのものが存在する．これを使うことで，GPIO を使用した I/O ボードの制御プログラムを JavaScript で書くことが可能となる．KUT-I/O ボードのような，個別の I/O ボードを制御するには，この API を組み合わせたプログラムを作る必要がある．

```
1       import RPI.GPIO as GPIO;
2
3       GPIO.setmode(GPIO.BORAD);
4       GPIO.setup(channel, GPIO.OUT);
5       GPIO.output(channel, GPIO.HIGH);
6       GPIO.cleanup();
```

図 2.1 Rpi.GPIO の GPIO 制御処理

2.2 Python で RaspberryPi を制御するライブラリ

リストに挙げる 3 つのライブラリは，RaspberryPi の制御を python で書けるようにするライブラリである．

- WiringPi
- Rpi.GPIO
- pigpio

WiringPi は RaspberryPi で使用されている BCM2835 などの SoC デバイスの GPIO へアクセスが可能なライブラリである [4]．WiringPi は GPIO へのアクセスに特化している．Python だけでなく，C 言語や C++，シェルスクリプトなどにも対応している．入力割り込みには対応していない．

Rpi.GPIO では簡単に割り込み入力ができる [5]．Rpi.GPIO は Python 以外の言語には対応していない．しかし，GPIO の入出力制御が非常に簡単である．Rpi.GPIO では，例えば図 2.1 に示すプログラムのように行う．4 行目で GPIO ピンを出力状態にして，5 行目で 1 を出力させている．

pigpio は RaspberryPi の GPIO を制御する API を豊富に持つライブラリである [6]．しかし，pigpio を使用する場合は，事前に pigpiod というデーモンを立ち上げる必要がある．

第 3 章

eJSVM

本章では，本研究室で開発している組込みシステム用 JavaScript 仮想機械である eJSVM について説明する．

3.1 eJSVM の実行形式

eJSVM は，バイトコードと呼ばれる VM 専用の命令の列を逐次実行するインタプリタの形式をとっている．そのため，eJSVM 上でプログラムを実行するためには，まず JavaScript のプログラムを専用のコンパイラによってバイトコード列にコンパイルする．これによって得られたバイトコード列を eJSVM に入力することでプログラムを実行する．

3.2 eJSVM が扱うデータの形式

JavaScript は整数，浮動小数点数，真偽値，オブジェクトなど様々なデータ型を扱う．C 言語で記述された eJSVM は，これらのデータ型の値を JSValue 型という一つの C 言語の型で扱っている．図 3.1 に eJSVM 内で扱う基本的なデータ形式の概略図を示す．JSValue 型の値は，基本的にはヒープ中にあるオブジェクトを指すポインタである．このオブジェクトのボディ部にデータの本体が存在する．オブジェクトのヘッダ部には，そのオブジェクトが JavaScript のどのデータ型かを示すための型情報が含まれている．なお，一部のデータ型は，JSValue 型の値がポインタである代わりに即値としてデータを埋め込んでいる．

eJSVM では，以上のようなデータの形式をとっているため，C 言語で実装されているバイトコード命令や組込み関数は値を直接扱うことはできない．JSValue 型の値を，必要に応

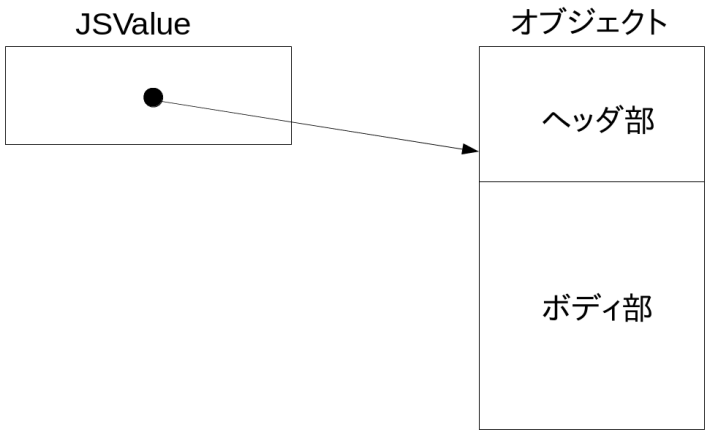

図 3.1　データ形式の概略図

じて C 言語で扱える型の値に変換する必要がある．例えば，倍精度浮動小数を使って計算を行うときは，JSvalue 型から C 言語の double 型に変換する必要がある．また，必要に応じて逆に C 言語の double 型から JSValue 型に変換する必要もある．このような変換は，VM の実装中で必要になることが多いので，変換用の関数が用意されている．例えば，浮動小数点数である JSValue 型の値から C 言語の double 型の値に変換する関数 `flonum_to_double` は，引数に浮動小数点数の値を表現する JSValue 型の値を一つとり，その変換結果として double 型の値を返す．`double_to_flonum` は，double 型の値を引数にとり，それを表す JSValue 型の値を返す．本研究で開発するライブラリは，整数と浮動小数点数を扱う．これらの値をもつ JSValue 型の値と C 言語の型とを相互に変換する関数を表 3.1 に示す．

変換するデータ型	関数名
JSValue(整数) → cint	`fixnum_to_cint`
cint → JSValue(整数)	`cint_to_fixnum`
JSValue(浮動小数点数) → double	`flonum_to_double`
double → JSValue(浮動小数点数)	`double_to_flonum`

表 3.1　本研究で扱う相互に変換する関数

第 4 章

ライブラリの設計

本研究では，高知工科大学情報学群で開講されている講義に使われる授業用の KUT-I/O ボードのすべての装置を簡単に制御できるようにするため，JavaScript で使用できるライブラリ関数を作成した．表 4.1 は，本研究で作成した全てのライブラリ関数である．

本章ではまず，ライブラリの要件を説明する．そして，KUT-I/O ボードのそれぞれの装置を簡単に使うためのライブラリの設計について説明する．

表 4.1　作成したライブラリ関数一覧

関数名	動作説明
`light()`	I/O ボード上にある LED を点灯する．
`lightOff()`	I/O ボード上にある LED を消灯する．
`col`m `(n1, ... ,n8)`	引数に指定したディスプレイの m 行目を点灯する．m 列目以外は消灯させる．引数は可変長引数であり，点灯させたい列番号を列挙する．
`button(n)`	スイッチの情報を取得する．`true` か `false` を返す．引数 `n` には使用するボタンを指定する．
`timer()`	システムタイマの値を取得する．
`sound(key, time)`	引数に指定した音をスピーカーから出す．`key` には音階，`time` には音を伸ばす時間を指定する．

4.1　ライブラリの要件

KUT-I/O ボードの全ての装置は，RaspberryPi の装置である GPIO ピンに繋がっている．GPIO はメモリマップト I/O になっている．この制御をアセンブリ言語で書く場合は GPIO を制御するための特定のアドレスやビットを意識する必要がある．そうすると作業時間が大幅に増加する上にバグが混入しやすい．その結果，短時間で試作できるプログラムの数が限られる．

そこで本研究で作成するライブラリ関数は，以下の要件を満たすようにする．

1. ライブラリ関数を呼び出した時点で KUT-I/O ボードの対応する装置を初期化し制御を可能にする．
2. 全ての装置を制御できるようにする．

4.2　ライブラリの設計

本研究で開発するライブラリは，組込みオブジェクト `Raspi` と，そのメソッドとして実装する．KUT-I/O ボードの全てをカバーするには，LED，ディスプレイ，スイッチ，スピーカーを制御するライブラリ関数が必要である．以下では，それぞれのライブラリ関数の設計を述べる．

4.2.1　LED

LED には点灯と消灯が必要である．そのため本ライブラリでは，LED を点灯させる関数と消灯させる関数を用意する．関数は `Raspi.light()` と `Raspi.lightOff()` とする．また，LED を点灯，消灯させるという単純な制御を行う関数になるため，これらの関数は引数を取らず，必ず `Undefined` を返す．

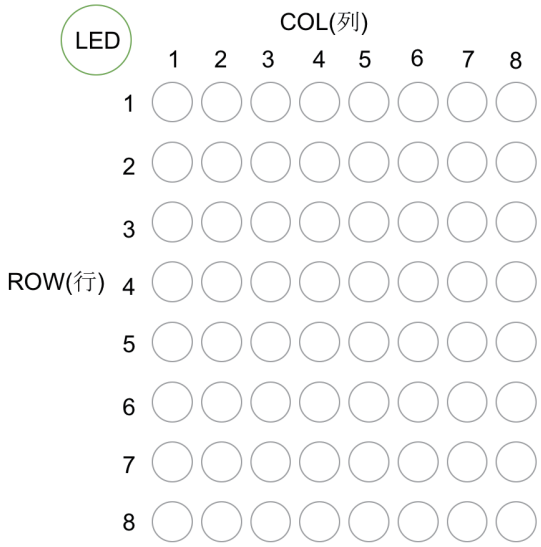

図 4.1　KUT-I/O ボードのディスプレイ

COL	ROW	結果
0	0	消灯
0	1	消灯
1	0	点灯
1	1	消灯

表 4.2　ディスプレイを点灯させる行と列に送る電位の対応表

4.2.2　ディスプレイ (8×8 ドットマトリクス LED)

ディスプレイは図 4.1 のように，LED が 8 行 8 列に並んだ電子部品である．LED を点灯させるためには，8 本の行 (ROW) を制御する信号線と 8 本の列 (COL) を制御する信号線を使う．各 LED を点灯させるか消灯させるかは，それぞれの信号線を制御することで可能になる．LED は列の信号線に接続された電位が，行の信号線に接続された電位よりも十分高くなった時に点灯する．電位は，列と行に対応する GPIO に 1 を出力すると高電位になり，0 を出力すると低電位となる．表 4.2 のような組み合わせでディスプレイの制御が可能となる．

表 4.2 のような組み合わせの電位を，それぞれの GPIO に送る必要がある．しかし，ディスプレイを点灯させるたびに行と列の GPIO を制御するのはユーザにとって負担となる．つまり，ディスプレイの点灯を簡単にするには，行と列へのそれぞれの電位をユーザに意識させる必要がない設計にする必要がある．そのため，本ライブラリでは，特定の一列のみを制御する関数を 8 つ用意する．関数はそれぞれ `Raspi.col1()`，`col2()`，`...`　`,col8()` とする．列と行の GPIO の制御は組込み関数の内部で行うようにする．この関数の引数には，その列中の行数に対応する値を記述するだけで，値の行を点灯できるようにする．引数には整数型の値に限る．また，引数は 1 個から 8 個の範囲で可変個とする．返り値は `Undefined` とする．

4.2.3　スイッチ

KUT-I/O ボードの装置の一つである，スイッチを制御する関数の設計を行った．スイッチは KUT-I/O ボード上に 4 つ設置されており，それぞれのスイッチの GPIO は異なっている．スイッチを押すことで電位の変化が起こり，その変化によって押したか押されていないかを判別する．本ライブラリでは，スイッチが押されている状態であれば true を返し，押されていない状態では false を返す関数を提供するようにした．関数は `Raspi.button()` とする．この関数は引数には，どのスイッチの値を返すかを選択するための数値をとる．引数に取りうる値は 1 から 4 までの数値で，図 4.2 のボタンに対応した値を引数に書くと，対応したスイッチの状態を論理型の値で返す．

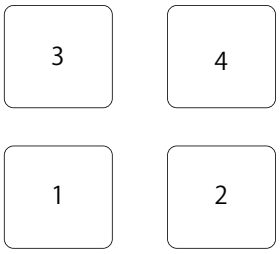

図 4.2　KUT-I/O ボードのスイッチに対応した値

4.2.4　タイマ

RaspberryPi に内蔵してあるシステムタイマを使うことによって正確な時間を計測することができる．RaspberryPi のシステムタイマは，64 ビットのカウンタレジスタを持っており，電源を入れたと同時に 1 マイクロ秒毎にカウンタの値が 1 ずつ増える．例えば，1 秒を計りたい場合はカウンタの値の下から 21 ビット目 (bit20) を見る．カウンタは 1 マイクロ秒で 1 増加するので，bit20 は 2^{19} マイクロ秒 (=512 × 2048 マイクロ秒) 毎に変化する．つまり bit20 が変化するのは 1 秒毎であるため，bit20 の変化を見ることで 1 秒の計測が可能である．本ライブラリでは，このカウンタの値を返す関数を提供する．関数は `Raspi.timer()` とする．この関数は値を読み出すだけなので，引数を取らない．返り値は，関数が呼び出された時点での RaspberryPi のシステムタイマの値である．返り値の型は実数型の値とする．

4.2.5　スピーカー

スピーカーは KUT-I/O ボード上に 1 つだけ設置されている．スピーカーから音を鳴らすためには，音の周波数を設定して出力することで可能となる．周波数を設定するには RaspberryPi に搭載されている PWM(Pulse Width Modulator) を使う．本ライブラリでは，第一引数に音階を表す値をとり，それに対応した周波数の音が鳴り，第二引数に音の長さを表す値をとり，それに対応した時間，音を鳴らす関数を提供する．関数は `Raspi.sound()` とする．

第 5 章

ライブラリの実装

4 章の設計を基に KUT-I/O ボードを制御するライブラリ関数の実装を行った．本章では，KUT-I/O ボードのそれぞれの装置を制御する関数の定数を説明する．

5.1 LED を制御する API の実装

LED を点灯する関数 `light()` の処理の流れを以下に示す．

1. LED の GPIO を制御するために，`mmap` システムコールを使い GPIO の制御レジスタの番地を eJSVM プロセスにマップする．GPIO のベースアドレスとなる番地は 0x3f200000 番地である [2].
2. LED を制御する GPIO の 10 番ピン (以下，GPIO10) を出力用に設定する．出力用にするには，GPIO10 を制御している番地である 0x3f200004 の番地へ 2 進数の 001 を書き込む．
3. GPIO10 から 1 を出力させ LED を点灯させる．GPIO10 に 1 を出力するには，0x3f20001c から始まる数ワードの，GPIO のポート番号 (この関数ならば 10) の対応するビットに 1 を書き込む．これで LED が点灯される．
4. `munmap` システムコールでマップを解除する．

LED を消灯させる関数 `lightOff()` の処理は `light()` の処理の流れの 3 番目で 0 を出力するように変えるだけである．GPIO10 に 0 を出力するには，0x3f200028 から始まる数ワード中の対応するビットを 1 にする．

列番号 (m)	GPIO ポート番号
1	27
2	8
3	25
4	23
5	24
6	22
7	17
8	4

表 5.1　ディスプレイの列番号と GPIO ポート番号の対応表

```
1        Raspi.col2(3, 5);
```

図 5.1　ディスプレイを制御する関数を使ったプログラム例

5.2　ディスプレイ (8 × 8 ドットマトリクス LED)

ディスプレイの m 列目を点灯させる関数 `col`m`(n1,n2,...,n8)` は，まず全ての引数 (`n1`，`n2`，...，`n8`) を C 言語の値の表現に変換する．この引数がとる値は整数型で，C 言語の int 型へ変換する．次に，m 列のピンが接続された GPIO を制御する．また引数で指定した行のピンが接続された GPIO も制御する．行番号と列番号それぞれのピンと GPIO の対応を表 5.1，表 5.2 に示す．すべてのピンを出力に設定したら，点灯させたい列の GPIO に 1 を，それ以外の GPIO に 0 を書き込む．そして，行の GPIO に 0 を，それ以外の GPIO に 1 を書き込む．そうすることでディスプレイの点灯ができる．

図 5.1 は，ディスプレイを図 5.2 のようにするプログラムである．図 5.2 の塗りつぶされた箇所は点灯している状態，それ以外は消灯している状態を示す．以下に，このプログラムを例として処理の流れを示す．

行番号	GPIO ポート番号
1	14
2	15
3	21
4	18
5	12
6	20
7	7
8	16

表 5.2　ディスプレイの行番号と GPIO ポート番号の対応表

1. 引数の値を変換する．組込み関数の実行は C 言語で行うため，`fixnum_to_cint` という変換関数を使い JSValue 型の値を C 言語の int 型へ変換する．
2. ディスプレイの GPIO を制御するために，`mmap` システムコールを使い GPIO の制御レジスタの番地を eJSVM プロセスにマップする．
3. ディスプレイの列を制御する GPIO と行を制御する GPIO を出力用に設定する．
4. ディスプレイの 2 列目を制御する GPIO のポート番号 18(GPIO8) に 1 を出力させ，それ以外は 0 を出力させる．GPIO8 に 1 を出力するには，0x3f20001c から始まる数ワード中の GPIO のポート番号の対応するビット (この例では 18 ビット) に 1 を書き込む．
5. 2 で変換した値に対応する行の GPIO ポート番号を求める．この例では，ポート 15 とポート 12 が求まる．
6. GPIO のポート番号 15(GPIO15) と 12(GPIO12) に 0 を出力させ，それ以外は 1 を出力させる．GPIO15 に 0 を出力するには，0x3f200028c から始まる数ワードの中の GPIO のポート番号に対応するビット (この例では 15 ビット) に 1 を書き込む．これでディスプレイの 2 列目の引数に設定した行が点灯する．
7. `munmap` システムコールでマップを解除する．

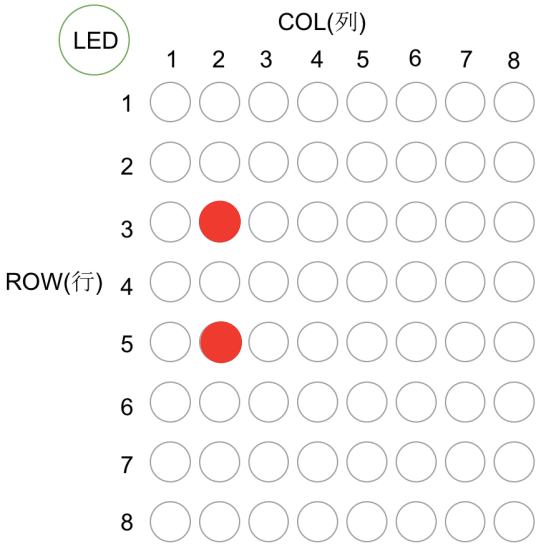

図 5.2　点灯するディスプレイの箇所

5.3　スイッチ

スイッチ n の状態を調べる関数 `button(n)` は，まず引数を C 言語の値の表現に変換する．この引数は整数型なので，C 言語の int 型へ変換する．次に，引数の値に対応したスイッチの GPIO を制御する．スイッチと GPIO の対応を表 5.3 に示す．スイッチが押されると，スイッチに対応している GPIO のポート番号のビットが変化する．スイッチが押されているとビットは 1 になっている．そうでなければ 0 になっている．スイッチが押されている場合 true を，押されていない場合は false を返す．この型は C 言語の型なので JSValue の boolean 型に変換する．`true_false` で C 言語から JSValue の boolean 型に変換し，返り値として関数に返す．

以下に，`Raspi.button(1)` という関数呼び出しを例として処理の流れを示す．

1. 引数の値を変換する．組込み関数の実行は C 言語で行うため，`fixnum_to_cint` という変換関数を使い JSValue 型の値を C 言語の int 型へ変換を行う．
2. スイッチの GPIO を制御するために，`mmap` システムコールを使い GPIO の制御レジスタの番地を eJSVM プロセスにマップする．

スイッチ番号	GPIO ポート番号
1	13
2	26
3	5
8	6

表 5.3　スイッチと GPIO ポート番号の対応表

3. どのスイッチを制御するかを手順 2 で変換した int 型によって決める．この例では引数が 1 のため，1 番スイッチを制御する．
4. スイッチが押されていると GPIO のポート番号 13(14 ビット目) が 1 となっている．そうでなければ 0 になっている．
5. 変化した場合は true を，しなかった場合は false を返す．
6. この boolean 型の値は C 言語の型なので，返す前に `true_false` によって JSValue の boolean 型に変換する．
7. `munmap` システムコールでマップを解除する．

5.4　タイマ

システムタイマを制御するためには，GPIO ではなくシステムタイマ専用の制御レジスタを制御する．タイマを制御する `timer()` の処理の流れを以下に示す．

1. `mmap` システムコールを使いシステムタイマの制御レジスタの番地を eJSVM プロセスにマップする．専用レジスタのベースアドレスとなる番地は 0x3f003000 である．
2. システムタイマから現在の値を取得する．まず，システムタイマのフリーランニングカウンタの値が入っているレジスタを制御する．
3. レジスタの番地は 0x3f003004 にある．このレジスタの値を取得する．
4. 取得した値は C 言語の double 型なので，関数に返す前に `flonum_to_double` によって

```
1       while(true) {
2               while((Raspi.timer() & (1 << 20)) == 1){
3                       Raspi.light();
4               }
5               while((Raspi.timer() & (1 << 20)) != 1){
6                       Raspi.lightOff();
7               }
8       }
```

図 5.3 タイマを使い LED を 1 秒ごとに点灯させるプログラム例

JSValue 型に変換する．

5. munmap システムコールでマップを解除する．

この関数を使い，1 秒ごとに LED を点灯させたい場合は図 5.3 のようなプログラムとなる．

5.5 スピーカー

sound(key, time) は，まず 2 つの引数を C 言語の値の表現に変換する．この引数は整数なので，C 言語の int 型へ変換する．次に，ライブラリ関数の第一引数 key には 220, 248, ... などの周波数を入れることで，対応した音階を鳴らすことが可能である．表 5.4 は音階と周波数の対応表である．次に，GPIO のポート番号 19(以下，GPIO19) と PWM のチャンネル 2 は配線を共有している．そのため，GPIO19 と PWM を接続するには 0x3f200004 の番地の 29 ビットと 27 ビットの間に 2 進数の 010 を書き込む．

PWM で音を鳴らすにはクロック信号を供給する必要がある．本研究では，講義に使われるクロックマネージャーの初期化プログラムを参考にした．クロックマネージャーのベースアドレスとなる番地は 0x3f1010004 とした．PWM で音を鳴らすために，M/S 転送モード

音階	周波数	音階	周波数	音階	周波数
3A	220	4A	440	5A	880
3B	248	4B	496	5B	992
3C	262	4C	523	5C	1048
3D	294	4D	587	5D	1174
3E	330	4E	659	5E	1318
3F	349	4F	698		
3G	392	4G	784		

表 5.4　音階と周波数の対応表

を使う．M/S 転送モードにするには CTL レジスタの PWEN ビットと MSEN ビットに 1 を設定する．

第 6 章

評価

本研究で作成したライブラリの有用性を確認するために二つの実験を行った．確認する点は以下の二つである．

- 本ライブラリが KUT-I/O ボードの全ての機能をカバーできるか．
- JavaScript と本ライブラリを使った KUT-I/O ボード制御プログラムは書きやすいか．

6.1 ライブラリの機能評価

本研究で作成したライブラリの機能評価として，KUT-I/O ボードの全ての装置を制御できるかを確認するための評価実験を行った．そのために，講義課題の演習課題を，本ライブラリを使って JavaScript で作成した．演習課題の内容は表 6.1 の通りである．

この実験の結果，すべての課題のプログラムを作成することができた．よって全ての装置を本ライブラリで制御できる事が確認できる．

6.2 KUT-I/O ボード制御プログラムの書きやすさ

本研究で作成した本ライブラリと JavaScript を使った場合の KUT-I/O ボード制御プログラムの書きやすさを評価するために，アセンブリ言語を使った場合の KUT-I/O ボード制御プログラムとの書きやすさを比較する評価実験を行った．本実験では，KUT-I/O ボードを制御する講義の単位を取得している学生 6 名を被験者とした．被験者にはそれぞれ JavaScript と本ライブラリを使った場合の KUT-I/O ボード制御プログラムとアセンブリ

プログラム名	プログラム内容
light.s	LED を点灯させるプログラム
blink.s	LED を点滅させるプログラム
switch.s	スイッチを押すと LED を点灯させるプログラム
23.s	(2,3) の箇所のディスプレイを点灯させるプログラム
uzu.s	ディスプレイに渦を表示させるプログラム
count.s	00 から 99 まで繰り替えしディスプレイに表示させるプログラム
bit19.s	0.5 秒ごとに LED を点滅させるプログラム
ra.s	ラの音を鳴らすプログラム
doremi.s	ドレミファソラシドを鳴らすプログラム

表 6.1　アセンブリ言語で書いた KUT-I/O ボード制御プログラム

言語を使った場合の KUT-I/O ボード制御プログラムを一つずつ作成させた．作成させる順番は，最初にアセンブリ言語で作成させ，次に JavaScript で作成させた．作成させたプログラムは講義中の演習課題である KUT-I/O ボードのディスプレイに渦を表示するプログラムである．本実験での評価項目は以下の 3 つである．

- プログラムの作成時間
- プログラムを修正して実行した回数 (修正回数)
- プログラムの行数

それぞれの評価項目の結果の平均値をグラフにしたものが図 6.1，図 6.2，図 6.3 である．この結果から，全ての被験者がアセンブリ言語を使った場合の KUT-I/O ボード制御プログラムよりも，本ライブラリと JavaScript を使った場合の KUT-I/O ボード制御プログラムが短時間でプログラムを書けていることが確認できる．さらに試行回数の少なさから，誤りのないプログラムが書きやすいことが確認できる．つまり，本ライブラリと JavaScript を使った場合の KUT-I/O ボード制御プログラムの方がアセンブリ言語を使った場合の KUT-I/O

制御プログラムよりも書きやすいという事が確認できた．

また図 6.4 は被験者ごとに作成された 2 つのプログラムを比較したグラフである．このグラフより，被験者のアセンブリ言語を使ったプログラミング能力の差が明確に分かる．KUT-I/O ボードのディスプレイを制御するだけのプログラムであっても 400 行超えする被験者がいれば，100 行も書いていない被験者もいる．しかし，図 6.4 の本ライブラリを使った場合のデータから．本ライブラリを使えば，人の能力によらず KUT-I/O ボードのディスプレイを簡単に制御できる事が確認できた．また，6.1 節の評価実験の結果より，本ライブラリは KUT-I/O ボードの全ての装置を制御できるため，全ての I/O 制御は本ライブラリを使うことによって簡単に試作できると考察する．よって，本ライブラリを使うことで KUT-I/O ボード制御プログラムの試作が簡単になる．

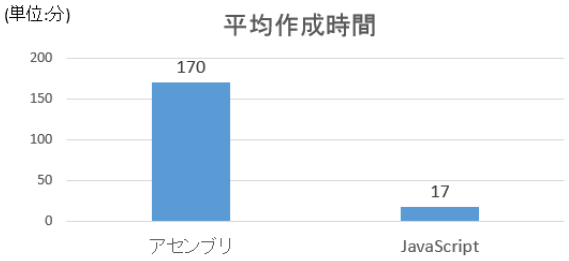

図 6.1　被験者が作成したプログラムの平均作成時間

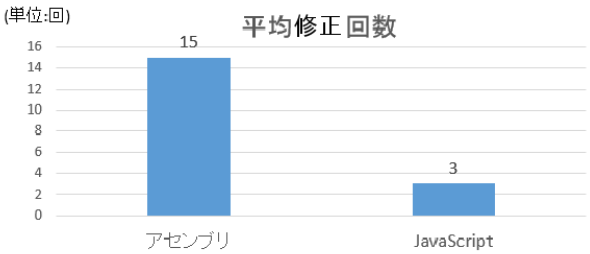

図 6.2　被験者が作成したプログラムの平均修正回数

6.2　KUT-I/O ボード制御プログラムの書きやすさ

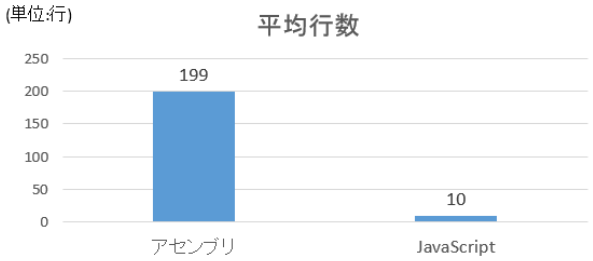

図 6.3　被験者が作成したプログラムの平均行数

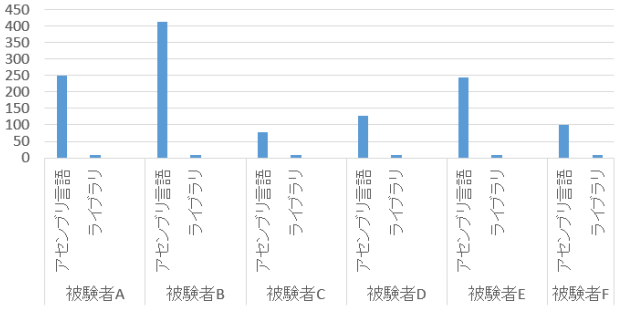

図 6.4　被験者ごとのプログラム行数

6.3　C 言語との比較

アセンブリ言語だけでなく，同じ低級言語である C 言語と，C 言語をサポートしたライブラリである WiringPi を使って，6.2 節で作成した渦を表示するプログラムを作成し，行数を比較した．結果をグラフにしたものが図 6.5 である．

C 言語と WiringPi を使うよりも JavaScript と本ライブラリを使った場合のプログラムが短い行数になっている．よって本ライブラリは，I/O 制御をするプログラムは C 言語とライブラリを使った場合よりも書きやすいことが確認できた．

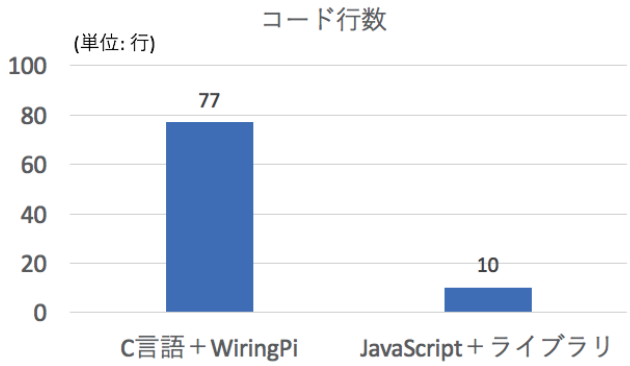

図 6.5　作成したプログラムの行数

第 7 章

おわりに

本研究では，KUT-I/O ボードを制御するプログラムを JavaScript を使って試作できるようにした．まず，KUT-I/O ボード制御プログラムを JavaScript で記述するために，eJSVM を JavaScript 処理系として用いた．次に，KUT-I/O ボードを制御するライブラリを開発した．

本ライブラリが KUT-I/O ボードの全ての装置を制御できるか確認した．その結果，KUT-I/O ボードのすべての装置を制御できることが確認できた．また，本ライブラリと JavaScript で書いた KUT-I/O ボード制御プログラムが，アセンブリ言語で書いた KUT-I/O 制御プログラムよりも書きやすいか評価した．評価の結果，本ライブラリを使用した場合のすべての被験者においてプログラムの作成時間が短縮され，開発中にプログラムを修正する回数も少なかった．さらに，開発されたプログラムも JavaScript と本ライブラリを使用した方が短くなった．これらの結果より，本研究で作成したライブラリを用いることで KUT-I/O ボードの試作が簡単になったことが確認できた．

謝辞

本研究を行うにあたり，一からご指導していただいた本学情報学群鵜川始陽准教授に深く感謝致します．また，副査を引き受けてくださった先生方にも感謝致します．さらに，評価実験の被験者を引き受けてくださった学生方にも感謝致します．最後に，研究室のメンバーをはじめ支えてくれた全ての人に感謝いたします．

参考文献

[1] 片岡 崇史，仮想機械の型ディスパッチャ自動生成ツールの設計と実装，平成 28 年度 学士学位論文，高知工科大学，2017

[2] Broadcom Corporation，BCM2835 ARM Peripherals，6.1-6.3, 2012

[3] jerryScript http://jerryscript.net/ 閲覧日 9 月 15 日

[4] Wiring Pi http://wiringpi.com/ 閲覧日 9 月 15 日

[5] SOURCEFORGE rapberry-gpio-python https://sourceforge.net/p/raspberry-gpio-python/wiki/BasicUsage/ 閲覧日 9 月 15 日

[6] pigpio library http://abyz.me.uk/rpi/pigpio/index.html 閲覧日 9 月 15 日