対話性と十分な実行速度を両立した組み込みマイコン向け開発環境の提案

前島 文香　山崎 徹郎　千葉 滋

本論文では，対話的に開発可能な組み込み向けの新しい言語である BlueScript とその処理系を提案する．BlueScript では Notebook 風のインターフェースで対話的に関数を実行したり追加したり修正したりできる．BlueScript の処理系はソースコードを C 言語に変換して，コンパイルと動的リンクを行うことで対話性と高い実行速度を両立している．さらに，コンパイル処理と動的リンクの大半をホストマシン上で行うことでマイクロコントローラのメモリ消費を抑えている．そのようなことをしないバーチャルマシンによる従来の実装方法では対話性は実行速度の低下を伴う．開発環境が対話的であれば細かい変更に対してビルドと書き込みをやり直す必要がないため試行錯誤による時間の浪費を削減できる上に BlueScript であれば実行速度の低下も小さい．BlueScript は TypeScript 風の構文を備えた言語であり，ソースコードは C に翻訳され，コンパイルされてから Bluetooth でマイクロコントローラに送信され実行される．性能評価として C や MicroPython [3] と BlueScript の間でマイクロベンチマークの実行にかかった時間を比較した結果を示す.

1　はじめに

マイクロコントローラの開発環境が対話的であれば，効率的に開発を行うことができる．マイクロコントローラの開発は実機を用いた細かい調整が必要になることが多い．例えばマイコンカーの速度や，デバイスのディスプレイ上のどの座標に文字を表示するのかは通常，実機で動かしながら微調整を行う．もしこのようなアプリケーションの開発を C 言語のようにビルドとフラッシュへの書き込みを行わないと実行できない開発環境で行うならば，試行錯誤のたびに決して短くない待ち時間が生じる．例えば ESP32 向けの C 言語のビルドシステム ESP-IDF では，微調整をするたびにビルドと書き込みのために 6–7 秒程度の待ち時間が生じる．もし開発環境が対話的であれば，このような待ち時間は生じないだろう．

このような利点がありながら，マイクロコントローラの開発において対話的な環境は広く使われていない．その理由は，対話的なインターフェースを提供する開発環境ではプログラムの実行速度が遅いためである．対話的な開発環境は通常バーチャルマシン (VM) を用いて実装されるが，VM はネイティブコードと比較して性能面で劣る．対話的な開発ではしばしば変数の値や関数の参照先を書き換えるため，VM による実装と相性がいい．実際に，代表的なマイクロコントローラ向けの対話的開発環境として MicroPython [3] や Espruino [7] が挙げられるが，これはどちらも VM によって計算を行う．しかしながら VM はバイトコードを解釈しながらプログラムを実行するため，追加のオーバーヘッドが生じる．計算速度が重視されるマイクロコントローラの開発ではこのようなオーバーヘッドは受け入れ難い．VM のオーバーヘッドを削減する技術として Just-In-Time コンパイラ (JIT) が知られているが，マイクロコントローラ上で利用可能なリソースは制限が強く，コードサイズが大きな JIT のマイクロコントローラへの搭載は困難である．

我々は，対話性と実行速度を両立したマイクロコン

---

* Proposal of development environment for embedded microcontrollers with both interactivity and sufficient execution speed.
This is an unrefereed paper. Copyrights belong to the Author(s).
Fumika Maejima, Tetsuro Yamazaki, Shigeru Chiba, 東京大学情報理工学部創造情報学専攻, Dept. of Information Science and Technology, The University of Tokyo.

トローラ向けの言語とその処理系である BlueScript を提案する．BlueScript は TypeScript 風の言語である．BlueScript の処理系は VM を使わず，ソースコードを C 言語に変換してコンパイルし動的リンクを行うことで，対話的でありながら高い実行時性能を達成している．さらに，コンパイルと動的リンクの大半を開発に使用中のホストマシン上で行い，マイクロコントローラには実行可能なバイナリのみを送信することで，処理系によるマイクロコントローラのメモリ消費を抑えている．本研究の主な貢献は既存の要素技術を新しい方法で組み合わせることで，対話的でありながら，高速なマイクロコントローラ向け言語処理系を作成した点である．

BlueScript の処理系を評価するために，我々は二つの実験を行った．一つ目の実験では BlueScript の実行速度を測るために，いくつかのマイクロベンチマークを用いて実行時間の計測を行った．二つ目の実験では BlueScript のプログラムの変更の容易さを評価するために，小さなプログラムの変更を反映するのにかかった時間を測定した．

2 対話的な環境と実行速度

マイクロコントローラ向けのアプリケーションを開発する際，対話的な開発環境がない場合には開発効率が悪い．小さなプログラムの変更に対しても，毎回時間のかかるビルドとフラッシュメモリへの書き込みの処理を行わなければならないためである．また，小さな変更を反映するために定義済みの関数を書き換えられる機能も重要である．

プログラム 1 は，あるマイコンカーを制御するためのプログラムである．このマイコンカーはマイクロコントローラによって制御されるおもちゃの車である．先頭に超音波距離センサーを搭載しており，目の前の障害物との距離を測定できる．また四つのタイヤを備えており，後輪は一つのモーターに接続されている．このモーターの回転数を調節することでマイコンカーの速度を調節することができる．プログラム 1 はマイコンカーを，目の前に障害物があると止め，なければ進める．開発者は停止距離とマイコンカーの速度を調節することでマイコンカーの動きを制御する．プログラム 1 において，停止距離は `BOADER_DISTANCE` の変数の値を変えることで調節が可能である．マイコンカーの速度は `go_straight` 関数を上書きし，`set_speed` の引数を変更することで調節可能である．停止距離とマイコンカーの速度の組み合わせが悪ければ障害物に衝突してしまう可能性がある．実際に停止する場所やマイコンカーの速度は，床からの摩擦やセンサーの測定精度によって変わるため，実機を動かしながら値の調整が必要となる．

マイクロコントローラのアプリケーションの開発は対話的な環境がない C 言語などで行われることが多い．しかし対話的な開発環境がない場合，このマイコンカーの開発は時間のかかるものとなる．開発者は停止距離やモーターの回転速度少し変えるためだけに，毎回時間のかかるプログラムのビルドとフラッシュメモリへの書き込みをやり直さなければならならず，開発の効率が落ちてしまう．

このような問題を回避するために，MicroPython などの対話的な開発環境を使うという選択肢がある．対話的な環境であれば，小さなプログラムの変更を容易に反映することができる．MicroPython の対話的な環境で開発を行う場合には，プログラム 1 の例で車のスピードを調整したいときは `go_straight` 関数を書き換えて実行ボタンを押すだけで良い．また，停止距離が短すぎると思った場合は `BOADER_DISTANCE` 変数だけを書き換えて実行ボタンを押せば良い．さらに，MicroPython では無線経由でソースコードを書き換えられる環境を提供しているため，動くマイコンカーとケーブルで繋がったパソコンを持ってマイコンカーを追いかけ回す必要もない．

このような利点がありながら，対話的なインターフェースを提供する開発環境は実行速度が遅いため限定的にしか使用されていない．対話的な開発環境では関数や変数の書き換えができることが望ましいが，そうした機能を提供するための開発環境はバーチャルマシン (VM) で実装されることが多い．VM による実装では，バイトコードを解釈しながら実行するため実行速度が遅くなる．実行速度を向上させるために JavaScript や Python などの処理系では Just-In-Time コンパイラ (JIT) を導入しているが，JIT は必

```
1  int MOTOR = 0;
2  float BOADER_DISTANCE = 5.0;
3
4  void go_straight() {
5    set_speed(MOTOR, 50);
6  }
7
8  void stop() {
9    set_speed(MOTOR, 0);
10 }
11
12 int app_main()
13 {
14   go_straight();
15   while (true) {
16     float distance = read_distance();
17     if (distance < BOADER_DISTANCE) {
18         stop();
19     } else {
20         go_straight(50);
21     }
22     delay_ms(500);
23   }
24 }
```

プログラム 1　マイコンカーを制御するプログラムの例

要なメモリ量が多すぎるためにマイクロコントローラ上に搭載するとアプリケーションが使用できるメモリを圧迫してしまう．

VM を使用しなくても対話的な開発環境を作ることはできる．分割コンパイルを行い，動的リンクをする方法である．この方法で実装された処理系はインタラクティブコンパイラと呼ばれ，例として C++ の REPL である Cling [8] や Rust の対話的な開発環境である Evcxr [2] などが挙げられる．この方法を使用すれば，対話的なインターフェースを持ちながら，実行速度の速いネイティブコードを生成することができる．しかしながら，インタラクティブコンパイラをマイクロコントローラ上に搭載することは現実的ではない．インタラクティブコンパイラは必要なメモリ量が多いため JIT と同様にマイクロコントローラ上に搭載するとアプリケーションが使用できるメモリを圧迫してしまう．また，最適化を含むコンパイルは性能の低いマイクロコントローラ上で行うと時間がかかりすぎてしまう．

図 1　BlueScript のインターフェース

3　BlueScript の提案

本章では対話的に開発可能でありながら，実行時性能の高い組み込み向けの新しい言語である BlueScript を提案する．BlueScript は TypeScript 風の言語であり，ガベージコレクションによって自動的にメモリを管理する．また，漸進的型付を採用しており，コンパイル時と実行時の両方で型検査を行う．BlueScript の処理系は BlueScript のソースコードを C 言語に変換してからコンパイル，実行を行うため VM 上で実行するよりも高性能である．さらに，BlueScript の処理系は追加されたソースコードを動的リンクすることで対話的な実行を実現しているが，コンパイルと動的リンクの一部をホストマシン側で行いバイナリ転送することで，処理系によるメモリ消費を抑えている．

3.1　利用シナリオ

BlueScript は Jupyter Notebook 風のインターフェースを有しており，開発者は図 1 のような画面から実機を動かしながら対話的に開発を進めることができる．BlueScript を使用した開発では，開発者は以下のような手順で開発を進める．

1. シリアルケーブルを通してマイクロコントローラに BlueScript のランタイムのバイナリを書き込む．
2. ホストマシン上で BlueScript サーバーを立ち上

```
function blinkLed(n: integer) {
  for (let i = 0; i < n; i++) {
    ledOn();
    waitMs(1000);
    ledOff();
    waitMs(1000);
  }
}

blinkLed(5);
```

プログラム 2　BlueScript のソースコード例

げ，ブラウザから`http://localhost:3000/repl`にアクセスする．

3. 開いたブラウザのページ上 (図 1) で BlueScript プログラムを書き，セルの左側の実行ボタンを押す．このようにすることで，追加したプログラムをマイクロコントローラ上で実行することができる．
4. マイクロコントローラの実際の動きを見ながら，一番下のセルにプログラムを追加して実行することを繰り返す．

対話的な開発は Bluetooth 経由で行うため，最初に BlueScript 本体のバイナリを書き込んだ後はシリアルケーブルは必要ない．

3.2　BlueScript

我々は組み込みマイクロコントローラ向けのプログラミング言語として BlueScript と呼ぶ言語を新たに開発している．エディタなどの既存の開発環境をなるべく流用できるよう，BlueScript は TypeScript の主要な構文をそのまま流用している言語である．プログラム 2 に LED を点滅させる BlueScript ソースコードの例を示す．BlueScript では実行速度の向上のため 32 bit `integer` 型と 32 bit `float` 型を静的に区別する．また，漸進的型付を行っており，`integer` や `float`，`string` などの他に任意の型の値を格納できる `any` 型を提供している．全ての `any` 型の値は 2 bit でタグ付された 32 bit で表現される．そのため，`any` 型に整数や浮動小数点数を変換した場合の精度は 30 bit に落ちる．`function` キーワードによる関数定義が可能で，再定義もできる．一方でプロトタイプや `eval` などの過度に動的な機能の大半は提供しない．また，現状では関数クロージャーやクラスなどには対応していない．BlueScript はマーク&スイープ方式のガベージコレクタを採用している．

3.3　BlueScript の処理系の実装

BlueScript コンパイル処理の大半をホストマシンにオフロードすることで，マイクロコントローラのメモリを圧迫することなく対話性と実行速度の両立を実現している．ホストマシン上では構文解析から動的リンクまでを行い，ゲストマシンであるマイクロコントローラ上で行う処理はバイナリの配置のみを行う．

3.1 節の手順 1 で開発者が最初に書き込むバイナリにはランタイム関数やネイティブ関数をコンパイルしたものが含まれている．ランタイム関数とは，関数の書き換えや自動メモリ管理，動的な配列要素の追加などの動的要素を支援するための関数である．ネイティブ関数とは，ユーザーが呼び出し可能なハードウェアを制御するための基本的な関数である．

3.1 節の手順 3 で開発者がソースコードを追加して実行ボタンを押すと，次の 4 つの処理がホストマシンのローカルに立ち上げられたサーバー内で行われる．(1) ユーザが入力した BlueScript のソースコードの構文解析と型チェックを行う．(2) ソースコードの文字列を C 言語へ変換する．この際，BlueScript の動的要素をサポートするためにランタイム関数の呼び出しを埋め込む．(3) 変換した C 言語のプログラムを GCC を用いてコンパイルする．この際リンクは行わない．(4) あらかじめホストマシン上に保存しておいたランタイム関数やネイティブ関数，ユーザーが定義した関数のアドレスを用いて動的リンクを行う．動的リンクの詳細については次の節で述べる．生成されたバイナリは Bluetooth 経由でゲストマシンに送られる．

ゲストマシンで行う処理はバイナリの配置のみである．ゲストマシンではあらかじめ新しいバイナリを配置するための領域を確保しておく．ホストマシンからバイナリが送られてくると，ゲストマシンは受け取っ

```
#include "stdint.h"

uint32_t main() {
    blinkLed();
}
```

プログラム 3　C のソースコード

```
0000000c <main>:
0: 004136       entry a1, 32
3: 000025       call8
6: 420c         movi.n a2, 4
8: f01d         retw.n
```

プログラム 4　リンク前のバイナリ

```
0000000c <main>:
0: 004136       entry a1, 32
3: febb25       call8
6: 420c         movi.n a2, 4
8: f01d         retw.n
```

プログラム 5　リンク後のバイナリ

たバイナリをその領域の末尾に追加しエントリポイントの値を指定の変数に書き込む．その後セマフォを用いて追加されたプログラムを実行をするためのスレッドに通知すると，新しいプログラムが実行される．

3.4　動的リンク

この節では，ホストマシン上で動的リンクを行いマイクロコントローラ上で実行可能なバイナリを生成する方法について説明する．

プログラム 3 はコンパイル対象の C 言語のプログラムである．プログラム中に含まれている `blinkLed` 関数はあらかじめコンパイルされてマイクロコントローラ上に配置されているとする．

ゲストマシンに送信して配置した際に実行可能であるようなバイナリをこのプログラムから生成するためには，次のような手順で動的リンクを行う．(1) ホストマシン上で GCC を用いてプログラム 3 をコンパイルする．この際リンクは行わない．(2) GCC コマンドで生成された ELF 形式のファイルからプログラム 4 のような `main` 関数のバイナリを抜き出す．(3) `blinkLed` 関数の呼び出し箇所 (プログラム 4 の 3 行目) に `blinkLed` 関数が配置されているアドレスの相対アドレスを埋め込む．この相対アドレスは `main` 関数がマイクロコントローラ上で配置される位置と，call8 命令のオフセット (プログラム 4 では 3 byte), `blinkLed` がマイクロコントローラ上で配置されているアドレスから計算することができる．プログラム 4 に相対アドレスを埋め込んだ結果がプログラム 5 のバイナリである．(4) 次回バイナリが追加された時にリンクできるよう，`main` 関数のアドレスと `main` が使用した領域のサイズ (10 byte) をホストマシン上に記憶しておく．

このようにして，目的のバイナリであるプログラム 5 を得ることができる．得られたバイナリは 3.3 節で述べた方法でマイクロコントローラに送信され，実行される．

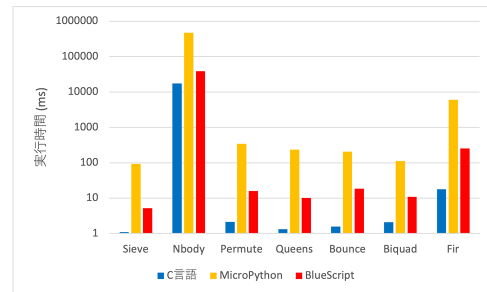

図 2　実行時間の比較

4　実験

本章では BlueScript の性能を評価するために行った二つの実験の結果を示す．一つ目の実験では，幾つかのマイクロベンチマークを用いて C 言語，MicroPython，BlueScript の実行速度を比較した．二つ目の実験では，BlueScript においてプログラムの変更の容易さを調べるために，プログラムを追加してから実行を開始するまでの時間を C 言語，MicroPython，BlueScript で比較した．どちらもまだ予備実験の段

表 1　プログラムの変更を反映するのにかかった時間

言語	セットアップ時間 (s)	反映にかかった時間 (s)
C 言語	43.387	7.086
MicroPython	93.980	0.234
BlueScript	32.463	0.435

階である．

4.1　実験環境

C 言語での実行時間の計測では，コンパイルに ESP-IDF の機能を用い，最適化オプションを O2 にして計測を行った．MicroPython の時間計測では公式ページから提供されているファームウェアの V1.19.1 を使用し，シリアルケーブルを用いてプログラムの書き込みを行った．BlueScript の時間計測では，BlueScript のプログラムを C 言語に変換したものをまとめてコンパイルして時間を計測した．この際コンパイルには C 言語での場合と同様に ESP-IDF の機能を用い，最適化オプションには O2 を用いた．

実験に用いたマイクロコントローラは M5Stack Fire に搭載されている ESP32-D0WDQ6 で，520 KB の RAM，16 MB の Flash，8 MB の PSRAM を搭載している．CPU の周波数は 240 MHz で，キャッシュサイズは 32 KB である．

4.2　実行時間の比較

C 言語，MicroPython，BlueScript の実行時間を比較する実験とその結果を示す．ベンチマークプログラムとして Marr らの Cross language compiler benchmarking [5] と ProgLangComp [6] のマイクロベンチマークを C 言語， MicroPython， BlueScript で書き直したものを使用した．まだ全てのベンチマークの実装が終わっていないため，今回は実装が終わったベンチマークでの実験結果のみを示す．

図 2 に各ベンチマークの実行時間を示す．縦軸は単位が ms の対数である．実行時間はベンチマークプログラムを 5 回実行し実行時間の平均を計算した．実行時間の分散は小さく，どのベンチマークでも 10 μs に収まっていた．BlueScript の実行時間は MicroPython での実行時間よりも大幅に短かった．MicroPython での実行時間が C 言語での実行時間の 27 から 339 倍であったのに対し，BlueScript での実行時間は C 言語での実行時間の 2 から 15 倍程度であった．BlueScript での実行時間が C 言語での実行時間よりも遅いのは，any 型値の unboxing や配列アクセスにおけるインデックスの検査が原因だと考えられる．

4.3　プログラムの変更にかかる時間の比較

プログラムの変更を反映するのにかかった時間を計測する実験とその結果を示す．この実験ではまず空のプロジェクトを作成してマイクロコントローラに書き込み，その後 5 行程度のプログラムを追加して再度マイクロコントローラに書き込んだ．この実験では空のプロジェクトを書き込むためのセットアップ時間と変更後のプログラムを書き込むためにかかった時間をそれぞれ計測した．

表 1 に各言語でセットアップにかかった時間とプログラムの変更を反映するのにかかった時間を示す．BlueScript では C 言語で実装した場合に比べて，反映にかかった時間が短かった．C 言語では反映に約 7 秒かかったが，BlueScript では 0.4 秒であった．この結果によって，BlueScript ではプログラムの書き換えが C 言語よりも容易に行えることがわかる．

また，BlueScript でプログラムの変更を反映するのにかかった時間は MicroPython で反映にかかった時間よりも約 0.2 秒長かった．この違いは，BlueScript がホストマシン上でプログラムをコンパイルすることによって生じたオーバーヘッドによるものであると考えられる．

5　関連研究

マイクロコントローラ上での高級言語の実行速度を上げる研究として StaticTypeScript [1] の研究が行われている．StaticTypeScript は，教育現場で使用されるマイクロコントローラの開発環境として，使いや

すく使用メモリ量が少なくて実行速度の速い開発環境の作成を目指した言語処理系である．StaticTypeScript も TypeScript 風の構文を有した言語であり，StaticTypeScript のソースコードはホストマシン上で C++に変換され一括コンパイルされてマイクロコントローラ上にロードされる．StaticTypeScript の研究も本研究と同様に実行速度の向上を目指してホストマシン上で TypScript 風の言語をコンパイルするが，対話的な環境をサポートしていないという点で本研究と異なっている．

我々の BlueScript のようなマイクロコントローラの開発を容易にするための研究には様々なものがあり，実行速度改善の研究のほかにもマイクロコントローラの開発を容易にするための研究として，リモートデバッグの研究も行われている．例えば Warduino [4] はマイクロコントローラ上で動く Wasm の VM であり，開発者が指定しておいたブレークポイントにたどり着くと，ホストマシンにスタックや変数の中身などの実行状態に関わる情報を送る．Warduino の研究は実機でのバグの特定を容易にはしているが，本研究が目的としているプログラムの変更が容易な開発環境は提供できていない．

6 まとめ

本論文では，対話的でありながら実行速度の速いマイクロコントローラ向けの開発環境である BlueScript を提案した．BlueScript の処理系はソースコードを C 言語に変換して，コンパイルと動的リンクを行うことで対話性と高い実行速度を両立している．また，コンパイル処理の大半をホストマシン上で行うことでマイクロコントローラのメモリ消費を抑えている．評価として C 言語，MicroPython，BlueScript の 3 言語でマイクロベンチマークの実行時間を測り，BlueScript での実行時間が MicroPython の場合と比べて短いことを示した．さらに，プログラムの変更を反映するまでにかかった時間を測定し，C 言語で開発する場合よりも BlueScript で開発した場合の方が短い時間で変更を反映できることを示した．

参考文献

[1] Ball, T., de Halleux, P., and Moskal, M.: Static TypeScript: An Implementation of a Static Compiler for the TypeScript Language, *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, MPLR 2019, Association for Computing Machinery, 2019, pp. 105–116.

[2] Evcxr authors: GitHub — evcxr/evcxr, `https://github.com/evcxr/evcxr`, 2022.

[3] George Robotics Ltd.: MicroPython, `https://micropython.org`, 2014.

[4] Gurdeep Singh, R. and Scholliers, C.: WARDuino: A Dynamic WebAssembly Virtual Machine for Programming Microcontrollers, *Proceedings of the 16th ACM SIGPLAN International Conference on Managed Programming Languages and Runtimes*, MPLR 2019, Association for Computing Machinery, 2019, pp. 27–36.

[5] Marr, S., Daloze, B., and Mössenböck, H.: Cross-Language Compiler Benchmarking: Are We Fast Yet?, *Proceedings of the 12th Symposium on Dynamic Languages*, DLS 2016, Association for Computing Machinery, 2016, pp. 120–131.

[6] Plauska, I., Liutkevičius, A., and Janavičiūtė, A.: Performance Evaluation of C/C++, MicroPython, Rust and TinyGo Programming Languages on ESP32 Microcontroller, *Electronics*, Vol. 12, No. 1(2023).

[7] Pur3 Ltd.: Espruino, `https://www.espruino.com`, 2017.

[8] Team, R.: GitHub — root-project/cling, `https://github.com/root-project/cling`, 2007.