関数型リアクティブプログラミングによる組み込みシステム開発の研究

Nakano, Fumiaki / 中野, 史彬

(出版者 / Publisher)
法政大学大学院情報科学研究科
(雑誌名 / Journal or Publication Title)
法政大学大学院紀要. 情報科学研究科編 / 法政大学大学院紀要. 情報科学研究科編
(巻 / Volume)
12
(開始ページ / Start Page)
1
(終了ページ / End Page)
6
(発行年 / Year)
2017-03-31
(URL)
https://doi.org/10.15002/00014409

関数型リアクティブプログラミングによる組み込みシステム開発の研究

Research on Development of Embedded Systems Using Functional Reactive Programming

中野　史彬
Fumiaki Nakano
法政大学大学院情報科学研究科情報科学専攻
E-mail: 15t0012@cis.k.hosei.ac.jp

Abstract

Functional languages are easy to detect bugs in a program at early stages and the program is unlikely to cause unexpected behavior, mainly because reference transparency is maintained, therefore, it is useful for assembling a high quality system. However, writing programs in a functional language tends to be hard when the system needs to acquire information from the outside, updating values by processing on that information. Functional reactive programming (FRP) is gaining attention as a framework to declaratively describe such processing. FRP is a very useful programming paradigm for describing operations that requires time handling such as physical operations, games, animation, parallel processing and other processing in real world domains. However, FRP is difficult to understand and cannot be handled easily. It takes time to actually perform coding. Therefore, in this research, we have implemented a library for embedded systems that makes FRP easier to handle. This makes it possible to improve code size reduction, readability, and ease of use. In this experiment, this library was able to reduce in code amount. Readability and ease of use were improved by the functions provided by the library.

1.　まえがき

自動車やロボット等の制御システムは近年複雑化してきており，テストやデバッグ等が難しくなっている．また，自動化が進むにつれ安全性がより求められるようになり，このようなシステムではバグによる予期しない動作を起こさないよう細心の注意を払う必要がある．

一方で，関数型言語は言語の特徴からバグの早期発見がしやすく，また参照透過性が維持されているためプログラムが予期しない動作を起こしにくい．そのため質の高いシステムを組むのに役立つと考えられるが，反面外部から情報を取得し，その情報に対し処理を行い値の更新をすることは参照透過性の影響で不得意である．

外部からの情報の取得や値の更新，時間に対する処理などを宣言的に記述する枠組みとして，関数型リアクティブプログラミング(Functional Reactive Programming 以下 FRP)が注目されている．

FRP は物理的な演算[1]やゲーム[2]，アニメーションなど時間を扱う操作や並列処理などを行う上で，非常に有用なプログラミングパラダイムである．しかし， FRP を理解することは難しく簡単に扱えないため，実際にコーディングを行えるようになるまで時間がかかってしまう．

そこで本研究では，より FRP を扱いやすくした組み込みシステム開発の研究を行う．本研究は FRP を利用できる環境を構築し，その上で FRP をより扱いやすくすることを目的とする．これにより，コード量の削減や可読性，利用しやすさを向上させることができる．ただし，本システムを利用する上でハードウェア実装の面でいくつかの制約がかかることがあるが，これによって表現力に影響が出ることはない (6 節後述).

本システムの有用性を示すために本研究では車両型ロボットでの様々な状況を想定したシステムを組み，そのシステム構築に対してコード量の削減が行われているか，可読性や使いやすさの向上ができているかを本システムの有無や FRP を利用しない場合と比較し考察する．

以下、第 2 節では本研究で利用する技術を紹介し，第 3 節では本システムの概要，第 4 節では組み込みシステムの実装方法を述べる．第 5 節で本システムの有用性を示すための実験方法を提示し，続く第 6 節では実験に対する考察を行う．最後の第 7 節でむすびとする．

2.　関連技術，関連研究

本研究では組み込みシステム向けの制御言語としての FRP を実現するためにハードウェアとして Raspberry Pi, LEGO Mindstorms，BrickPi を用いる．また FRP を利用するにあたり，汎用 FRP 向け Haskell ライブラリの Netwire を利用する．それぞれの関係は図 1となる．

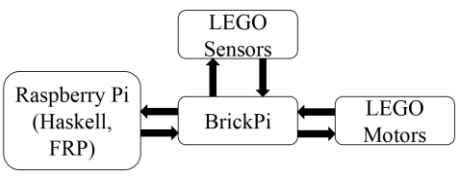

図 1 機器の関係

Supervisor: Prof. Akira Sasaki

Raspberry Pi から命令を出し，その命令やコマンドを BrickPi に渡しモータやセンサに受け渡す．命令によってはセンサから値の取得を行い Raspberry Pi に渡し処理を行う．これが一連の流れとなる．また，2 節で述べる機器を組み合わせたロボットが図 2となる．

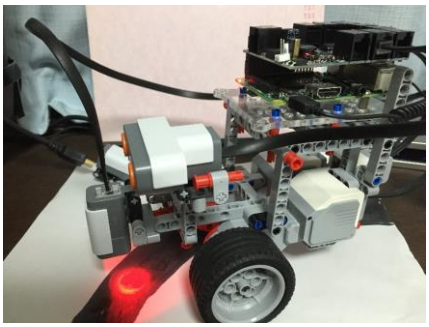

図 2 車両型ロボット

2.1. 関連技術

2.1.1. 関数型リアクティブプログラミング

関数型リアクティブプログラミング(Functional Reactive Programming)とは，関数型言語の機能を維持しつつ副作用のある操作も利用できるようにしたものである．これにより，時間とともに変化する値の操作が利用しやすくなる．FRP の実例として，表計算ソフトがあげられる．表計算ソフト上でセル A1 に「=B1+C1」のように記述すると，セル B1 やセル C1 の値が変更されたとき，セル A1 に出力される値が固定されるのではなく，動的に書き換えられる．このように値が変更されるとそれに合わせて他の値も動的に変更されることが特徴である．

次に本研究で利用するような例として，各センサの値を取得しその値の和を求めるものを考える(図 3).

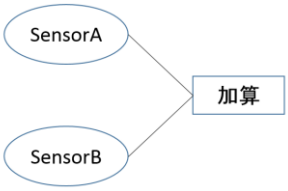

図 3 FRP での加算図

FRP を利用すると各センサの値が変更されたとき，そのタイミングに合わせて値の取得が行われ，その後加算処理を行い出力される値の更新が行われる．以上のような処理は，プログラマが明示的に記述せずとも任意のタイミングで値が自動的に更新されるため，手間を省くことができる．

FRP では，このような連続する時間に伴う変化を Behavior，離散的な時点で起こるシステムへの変化を Event という概念で表す．また，この 2 つのことをまとめて Signal と言う．図 3では，時間の経過が Behavior，センサの値が変化するタイミングが Event となる．FRP ではこれらの機能をもとにプログラムを記述していく．

2.1.2. Netwire [3]

Netwire は Haskell 用のライブラリで，汎用 FRP を提供する．Netwire は Wire という概念のもと設計を行う．Wire は入力に対し処理を行い，値を出力する関数である．この Wire は一般化されたオートマトンの Arrow と捉えられるため，これらを用いて多くのデザインパターンを表現できる．Wire は

Wire s e m a b

という形をしており，s はセッションタイム，e は抑制方法，m は内部のモナド，a は入力，b は出力を表している．基本的にこの型は値 a を受け取った後，処理を行い値 b の出力を行うものである．

また,本研究ではアロー記法という記法を用いる．これは，入出力などの関係を” 出力 <- 関数 -< 入力”のように矢印で表すものである．

Netwire は Yampa[4]をベースとしており，インターフェースとフレームワークの洗練化をし，ゲームサーバと知能ネットワークロボットの強化を目的としていた．しかし，Netwire は時間の間隔や Signal などを取り入れたため Yampa とは別物となっている．

本研究では，用いる機材の環境に対応しているバージョンが限定されているため Netwire-5.0.0 を用いる．

2.1.3. Raspberry Pi

Raspberry Pi は Raspberry Pi Foundation によって製造されているシングルボードコンピュータである．名刺サイズと小さく省電力で動作するが，性能が高いため最近では IoT(Internet of Thing)や個人サーバとして運用されていることが多い．本研究では後述の BrickPi と接続し，BrickPi から取得した値などを Raspberry Pi で処理し制御を行う．

2.1.4. BrickPi

BrickPi は DEXTER Industries 社が開発したロボットキットである．BrickPi は LEGO Mindstorms のモータやセンサ類をワンタッチで接続可能にするため，本研究ではセンサ類や制御機器とのインターフェースとして利用する．これにより，ハードウェア実装において時間がかかる工程を短縮でき，さらにハードウェア面での接続ミスなどのバグを減らすことができる．

本研究では DEXTER Industries 社が提供している BrickPi 制御用ライブラリを Haskell 処理系の GHC から利用できるようにし，その上で FRP を用いる．

2.1.5. LEGO Mindstorms

LEGO Mindstorms は LEGO 社が提供している教育向けロボットである．LEGO Mindstorms は様々なロボットを手軽に組み立てることができ，それらを LEGO Mindstorms のマイコンでプログラム・制御することができる．LEGO Mindstorms では Java や Python，ビジュアル言語などが対応している．そのため，プロトタイピング

にも向いていると考えられる．本研究で使用するものはカラーセンサと超音波センサ，モータなどである．カラーセンサは色の濃淡を検出するもので，超音波センサはセンサから超音波を発し，物体との距離を検出するものである．他に，音の大きさを検出するサウンドセンサや接触非接触の状態を認識するタッチセンサなどがある．

2.2. 関連研究

本研究の関連研究として Yampa[4], 澤田らの研究[5][6]について述べる．前者は FRP による実装に関する研究であり、シミュレーションとしてのロボットを扱う例が研究で挙げられている．後者は小規模組み込みシステムでの FRP を利用した DSL の実装である．本研究では前者とは実ロボットを対象としているので対象が異なり，後者とは FRP を利用した際の難しさを緩和することを目的とすること，組み込みシステム自体の規模の大きさが違うことなどが異なる．

Yampa[4]は FRP の 1 つでアニメーションライブラリである．Arrow を利用しプログラムを記述していく．文献[4]ではシムボット(ロボットシミュレータ)を扱い制御する．このシムボットは本研究で利用するロボットと同様な機能(モータ，センサ)を有している．

Cfrp[5]，Emfrp[6]は澤田らによって実装された小規模組み込みシステム向けのソフトウェアを記述するための DSL である．これらの研究では OS を持たないような小規模ハードウェアを対象とし，その上で動作するプログラムを FRP に基づく DSL によって開発することを可能としている．どちらの研究も実装した DSL が小規模システムにおいても有用であることを示している．

3. FRP による組み込みシステム向けライブラリの設計

本研究では，コード量の削減と可読性，使いやすさの向上を目的としている．本ライブラリはユーザが提供されるライブラリの内容を自由に選択できるものとし，ユーザ自身の熟練度にあった機能を提供する．ユーザの想定として，①初級者②中級者③熟練者を想定している．以降想定しているユーザとそのユーザに提供する機能について述べる．

3.1. 初級者向け機能

初級者向けの機能について述べる．ここでいう初級者とはほかのプログラミング言語を学んだことがある，または関数型言語に触れたことがある者とする．

提供する機能として

- 機器の初期化やセットアップ
- モータの制御
- センサの制御

があげられる．機器の初期化やセットアップは他に想定されているユーザと大きく異なり，すべての処理を提供している．本来であれば図 7のように自由に記述できるが，使う機器とポートの箇所を制限することにより 1 行で記述可能となる(図 4)．機器とポートの箇所を制限することにより自由度が失われると考えられるが，あくまで初級者向けということにし，ハードウェアとソフトウェアでの一定の型を保ったまま実装することを促進し，不用な処理を省くためである．

次に，モータの制御では LOGO のような言語を提供する．これにより，速度は一定となってしまうが手軽にモータの動かし方を制御することができる．

最後にセンサの制御だが，多くのセンサは閾値を定義し，その閾値に対して判定を行う．例えば，カラーセンサでは色の濃淡を判別するが白線と黒線を見極める閾値を設定する必要がある．しかし，本システムではあらかじめ設定してあるため，ユーザは閾値の設定をすることなく手軽に実装できる．

これらの機能を利用すると，単純な動作を行う車両型ロボットを手軽に実装することができる．

```
main1 = proc _ -> do
  _ <- initialAuto -< ()
  car  -< (White, Far)
sensors = proc _ -> do
  light <- getLightE -< ()
  returnA -< (light)
motors = proc light -> do
  case light of
    Black -> do
      _ <- straightMotor -< ()
      returnA -< ()
    White -> do
      _ <- leftMotor -< ()
        returnA -< ()
```

図 4 初級者ユーザが記述するコードの一部

3.2. 中級者向け機能

中級者向けの機能について述べる．ここでいう中級者とは 3.1.の初級者向けの機能だけでは満足できず，モータやポートの箇所，閾値などをより自由に設定したい者を対象とする．提供する機能は 3.1.と変わらない．ここではより自由に実装できることができる機能が提供される．例えば，3.1.ではポート箇所の制限や閾値の固定などがあった．ここでは，図 5のように扱うセンサとそのポート，モータのポートを自由に設定することができる．3.1 では 1 行で実装できていたものが複数行になるが，その分自由度の高い設定が可能となる．同様に閾値の設定やモータの動かし方なども自由に設定できるため，3.1.の機能だけ実装できなかったより高度なロボットの実装が可能となる．

```
initialize = proc _ -> do
 ---------- 中略 ----------
   True -> do
     _ <- initialSensor [(port_1,
                          type_sensor_light_on)] -< ()
     _ <- initialMotor [port_c, port_b] -< ()
---------- 中略 ----------
```

```
sensors = proc _ -> do
  light <- getLightT port_1 512 -< ()
  returnA -< light
motors = proc light -> do
  case light of
    Black -> do
      _ <- freeMotor [(port_c, 255), (port_b, 255)] -< ()
      returnA -< ()
    White -> do
      _ <- freeMotor [(port_c, 255), (port_b, 0)] -< ()
      returnA -< ()
```

図 5 中級者ユーザが記述するコードの一部

3.3. 熟練者向け機能

熟練者向けの機能について述べる．ここでいう熟練者とは，本研究で実装されたライブラリを自由自在に扱える者とする．熟練者になると提供された機能だけでは満足できない場合もある．そこで本ライブラリは拡張しやすい設計となっているため，ユーザが新たに機能を追加したり既存の機能を改良したりすることができる(図 6). 図 6のように import や data などが記述されているライブラリを土台とし，その上に初期化をするライブラリやセンサからの値を取得し判別するライブラリが存在する．その上にモータを制御するライブラリが存在し，初級者や中級者はこれらをまとめたライブラリを利用する．ライブラリの中に独立したライブラリがあるため，使用したいライブラリだけを利用することも可能である．

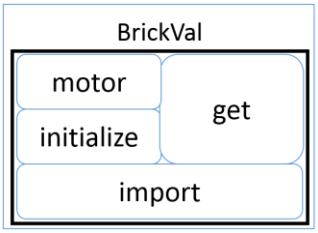

図 6 ライブラリの内部

4. FRP による組み込みシステムの実装

まず，プログラムが動く環境を構築するために，2 節で述べた機器を利用する．これらの資源は表 1のようになっている．

OS は，DEXTER Industries 社が提供している OS を利用する．この OS は，BrickPi を利用しやすいようにカスタマイズされた環境で，すでに Python や Scratch などが利用できるようになっている．これらを利用して BrickPi を制御することも可能である．本研究では Haskell を利用するため，この環境に Haskell を実行できるように Haskell 処理系の GHC(Glasgow Haskell Compiler)を導入する．また，FRP を利用するために Haskell のパッケージ管理システム cabal を利用し，その中に Netwire 5.0.0 をインストールする．

表 1 資源のバージョン

Model	Version
Raspberry Pi	Raspberry Pi 2 Model B
BrickPi	BrickPi
LEGO Mindstorms Motors	LEGO Mindstorms EV3
LEGO Mindstorms Sensors	LEGO Mindstorms NXT
OS	2016.12.04_Dexter_Industries_jessie (Raspbian)
GHC	7.6.3
Netwire	5.0.0

次に，Haskell で BrickPi が提供しているライブラリを利用できるようにする．そのために，ここでは FFI (Foreign Function Interface)を利用する．FFI とは特定のプログラミング言語上で，他のプログラミング言語で定義された関数やライブラリを利用するための機構のことである．本研究では C 言語で記述されている制御関数を Haskell で利用できるようにする．

4.1. 実装

組み込みシステムを実装するにあたり，3 節で述べたシステムを利用する．ユーザはこのシステムを使う上で主に 4 つのコードを記述することになる．

4.1.1. 初期化部

これは BrickPi の初期化やモータ，センサなどの設定を行う箇所である．ユーザはどのポートに何を接続するかを記述することでその機器を利用できるようになる．本システムを使うと図 7のように記述することができる．

initialSensor はポート番号と使うセンサの種類のペアをリストとして渡すことで利用できる．図 7では，カラーセンサを port_1 に，超音波センサを port_3 に接続する．initialMotor ではポート番号を指定するだけでモータの利用が可能となる．図 7では port_c と port_b を利用することを表す．initialSet では BrickPi などの初期設定を行ってくれる．

```
initialize = proc _ -> do
  result <- setup -< ()
  case result of
    True -> do
      _ <- initialSensor [(port_1, light_on),
                          (port_3, ultrasonic_cont)] -< ()
      _ <- initialMotor [port_c, port_b] -< ()
      _ <- initialSet -< ()
      returnA -< ()
  returnA -< ()
```

図 7 初期化部のサンプルコード

4.1.2. 統合部

統合部は車両型ロボットがどのようなセンサの情報を利用しモータを動かすか記述する箇所である．

```
car :: (HasTime t s) => Wire s () IO (Light, Dis) ()
car  = proc (light, dis) -> do
  _ <- motors -< (light, dis)
  _ <- chatter -< ()
  (light1, dis1) <- sensors -< ()
  car -< (light1, dis1)
```

図 8 統合部のサンプルコード

便宜上 car という関数名にしているが，本研究では車両型ロボットを主として扱ったためであり，命名はユーザに委ねられる．図 8ではカラーセンサと超音波センサの値によってロボットの振る舞いが変わる．カラーセンサと超音波センサの情報を motors 関数に渡すことでモータの動きを制御し，chatter 関数でチャタリング対策と同時にモータに制御した値を更新する．その後，センサの値を適宜更新する．car 関数を利用する際には，最初にダミーデータを与えなければならない．また，ここでは時間を扱う操作を行っているため関数に対し(HasTime t s)と明示しなければならない．

4.1.3. センサ部

これは，センサが値を取得する箇所である．センサごとに対応している関数が異なるのでユーザがどの関数を利用するか定義する必要がある．図 9は光センサと超音波センサの値を取得している．

```
sensors = proc _ -> do
   light <- getLight port_1 -< ()
   dis <- getDistance port_4 -< ()
  returnA -< (light, dis)
```

図 9 光と距離の値を取得するサンプルコード

4.1.4. アクチュエータ部

これは，モータの振る舞いを記述する箇所である．4.1.2.で呼び出され与えられたセンサの情報に対するモータの振る舞いを記述する．アクチュエータ部の記述の仕方は図 4の motors 関数のように記述する．

4.2. コンパイル

FRP を利用する際十分なスタック空間が必要となる．そのため，何もせずにコンパイルするとスタック不足となってしまうことがある．そこでコンパイルするときに空間を増設する必要がある(図 10)．これにより十分なスタック空間を確保することができる．

```
ghc ‐‐make LineTraceCar.hs BrickPi.o tick.o –rtsopts
sudo ./LineTraceCar +RTS –K100M
```

図 10 コンパイル時と実行時コマンド

5. 実験

本研究で提案するライブラリの有用性を実証するにあたり，以下の 2 つシステムを実装し，それぞれコード量の削減や可読性，利用しやすさの向上が行えているかを本ライブラリの有無で比較する．

i. ライントレースカーの実装
ii. 障害物を避け，目的地に停車する

i は線上を動く車両型ロボットである．線の濃淡を検出し，濃淡に合わせた振る舞いをする．ii は障害物を感知し，それらを避けながら目的地へ向かい目的地に到着すると停車処理を行うものである．i ではカラーセンサ，ii ではカラーセンサと超音波センサを使用する．また i, ii ともにモータを 2 つ使用する．コードの一部は図 11, 図 12のようになっている．

```
motors = proc (lightL, lightR) -> do
  case (lightL, lightR) of
    (Black, Black) -> do
      _ <- straightMotor -< ()
      returnA -< ()
    (Black, White) -> do
      _ <- rightMotor -< ()
      returnA -< ()
---------- 後略----------
```

図 11 i のコードの一部

図 11のコードは i のコードの一部である．この関数は，モータの振る舞いを表している．2 つのカラーセンサの値を受け取り左右どちらとも黒を検知すると両方のモータを回転させ，どちらかが白を検出していると黒線上に戻るように片方のモータを回転させる．両方とも白を検出している場合は左回転するように設定している．両方とも白線を検出している場合は他にも停止したり後退したりするなどユーザが自由に設定できるが本研究では左回転させることとする．

次に ii のコードの一部について述べる．

```
park :: (HasTime t s) => Wire s () IO () ()
park = after 2 . stopMotor <|> leftMotor
```

図 12 ii のコードの一部

図 12のコードは ii のコードの一部であり，停車するときの処理を表す．この関数は車両が停車位置についたときに呼び出され，車両がその場で一定秒間回転し，その後停車する処理を行う．after 関数は Netwire が提供している関数で”<|>”の右側の関数を指定秒間動作させ，その後左側の関数を実行し続ける．

6. 考察

実装したライブラリを利用し，5.i.と 5.ii.のシステムを本ライブラリの有無でコード量の比較を行った．また，5.i.のみ FRP を使わず Haskell のみで記述した場合とも比

較を行った．5.i.と 5.ii.のコード量の比較は表 2のようになった．表 2 の①は 3.1.の初級者向け機能，②は 3.2.の中級者向け機能の利用を表す.

表 2 コード量比較

	② 有	②有	FRP のみ	FRP 無
5.i.	41 行	52 行	86 行	73 行
5.ii.	47 行	59 行	111 行	--

関数型言語を用いて本研究で対象とするような組み込みシステムを実装するにあたりネックとなるのは関数の型の一致だが，これはライブラリ側で型が厳格に定義されているため，本ライブラリを利用することにより大部分を型推論で通すことができる．本ライブラリの機能を利用すると型推論が行いやすい形になっているため，型の一致を簡単にできる.

コード量の削減に関しては，本ライブラリを用いることにより，まず多くの import 部分を削除することができる．開発する上で必要なライブラリ群は本ライブラリに記述してあるため，ユーザが開発するたびに import 部分を書く手間が省かれる．次に，本ライブラリによって提供された部分はそのままコード量の圧縮のつながる．特に必ず呼び出す初期化の箇所やモータの簡単な動きなどがライブラリで提供されているため，コード量を多く削減することができる．また，コード量の削減によりコード全体が見やすく，かつわかりやすくなるため可読性の向上も行われている.

可読性はアロー記法を導入しているため，入出力の流れがわかりやすいものになっている.

本ライブラリの有無でコード量の比較を行う．本ライブラリを用いることにより，コード量に関して 5.i.では 50%以上の削減が見られ 5.ii.では 40%程度の削減が見られた．これは定型的に用いる箇所がライブラリで定義されているため，開発者が定義する必要がなくなった点が大きく作用している．①と②で比較を行うと①は②に比べて 20%程度削減していることがわかる．主に初期化部の差がそのまま表れている．加えて行数では測れないがそれぞれ閾値やポートの設定などで①側では大きく削減されておりコードを書く量が大きく減っている．FRP の有無で比較すると本ライブラリを利用する場合は①では 40%程度，②では 30%程度削減が見られた．しかし本ライブラリを利用しない場合と比べるとコード量が増えてしまっている．これは，FRP 特有の書き方をしたため 15%程度増えてしまったと考えられる.

FRP を利用することにより，副作用のような操作を利用する箇所と利用しない箇所を記述できるため，代入によるバグなどを気にせず記述でき，また代入などによるテストの記述を省略することも可能になる．これらの利点を組み込みシステムで簡単に扱えるのは非常に大きな利点であると言える.

本ライブラリのモータやセンサを制御する機能を利用するにあたって初級者はモータやセンサの接続箇所を指定のものに固定するなどの制約があるが，この制約は同時にハードウェアに対しての制約でもある。初級者がポートの接続などハードウェア面での実装において簡略化し，ソフトウェア，ハードウェアの両者を一定の型で実装することを促進するため，簡単に実装することができるようになる.

初級者が主に利用する機能のみで実装すると，機能外のシステムを実装する際にプログラムが記述しにくくなる可能性があるが，本ライブラリが抽象度の高い機能と同時に低レベルの機能も実装されているため，プログラムが記述しやすい.

7. むすび

本研究では関数型リアクティブプログラミングによる組み込みシステム開発の研究を行った．このライブラリにより，目的であるコード量の削減や可読性の向上，利用しやすさの向上が図れた．また，ユーザの熟練度に合わせて使用する機能を選べるため，幅広い層のユーザにも対応することができる.

今後の課題として，本ライブラリで提供される環境でのテストシステムの導入があげられる．センサ類が正常に動作するか適当な値をセンサに直接入れ，正常に動作するかテストする方法や，グラフィカルなモデルを作成しシミュレーションしながらテストする方法などが考えられる.

また，より汎用性を持たせるために BrickPi を利用せずに Raspberry Pi の GPIO を利用することも考えられる．その場合ハードウェアの実装において BrickPi と比べて自由度が非常に高くなるため，ユーザ層に合わせた制約やテンプレートなどを提供することにより幅広いユーザに利用可能なライブラリとなる.

文 献

[1] Kenji Ohmori , “Development of Functional Reactive Programming Using an Incrementally Modular Abstraction Hierarchy” , The 5th International Conference on IT Convergence and Security 2015 in Kuala Lumpur, August 2015

[2] Antony Courtney, Henril Nilsson, John Peterson, “The Yampa Arcade”, Haskell '03 Proceedings of the 2003 ACM SIGPLAN workshop on Haskell , August 2003

[3] netwire: Functional reactive programming library , https://hackage.haskell.org/package/netwire , Online , accessed January 2017

[4] Paul Hudak, Antony Courtney, Henrik Nilsson, John Peterson, “Arrows, robots, and functional reactive programming.” In Johan Jeuring and Simon Peyton Jones, editors, Advanced Functional Programming, 4th International School 2002 volume 238 of Lecture Notes in Computer Science pages 159--187. Springer-Verlag, 2003.

[5] Kensuke Sawada, Kohei Suzuki, Takuo Watanabe, Towards Applications of FRP in Small-Scale Systems, IRICE Technical Report SS2015-1, May-2015

[6] Kensuke Sawada, Takuo Watanabe, Emfrp: A Functional Reactive Programming Language for Small-Scale Embedded Systems, MODULARITY Companion’16 in Spainm, March 2016