CPU＋FPGA プラットフォームのための Ruby ベースの開発環境
—Ruby で書ける CPU＋FPGA コデザイン環境 Mulvery—

1. 背景

IoT（Internet of Things）の分野では、センサの出力データやアクチュエータの制御信号など、様々な種類のストリーム型データが扱われる。IoT デバイス開発には汎用マイコンを用いることが多いが、このようなデータの処理はリアルタイム性の保証が必要な場合が多く、ストリームの数が増えるほど汎用マイコンの MPU 上での開発は複雑で困難なものとなる。そこで、FPGA（Field Programmable Gate Array）を導入することが考えられる（図 1）。FPGA は、目的とする処理を回路として実装することで並列に実行することが可能であり、処理能力や電力効率の面において注目されている。しかしながら、一般に FPGA における回路設計は多くの場合ハードウェア記述言語（Hardware Description Language, HDL）を用いて行うため、ソフトウェアベースのシステムと比べて開発のコストが大きくなる。

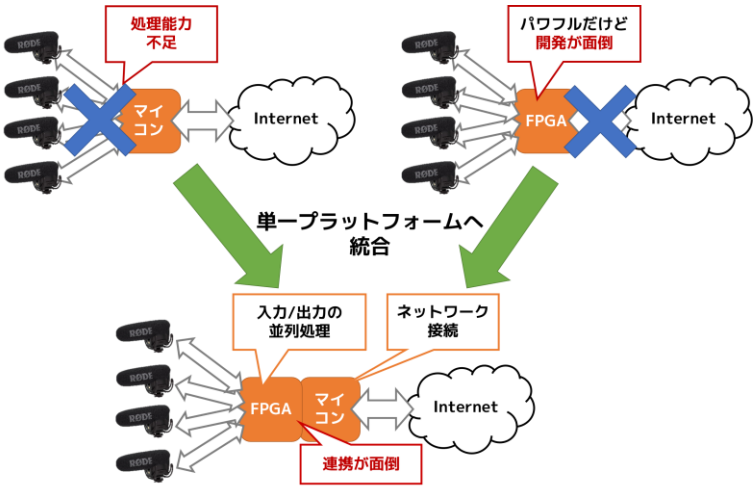

図 1 マイコンと FPGA の問題点と、それらを単一のプラットフォームへ統合した場合の問題点
（マイク画像：CC BY-SA https://jmcintyre.wikispaces.com/）

最近では、高位合成系と呼ばれる、高水準言語によるプログラムから等価なふるまいの回路を合成するツールがしばしば用いられる。外部モジュールとの通信ではタイミングが重視されるため、クロックを意識しない高位合成ではタイミング調整が困難であるという問題があり、IoT デバイス開発には適さない。

2. 目的

本プロジェクトでは、マイコンに FPGA を導入した IoT デバイスのさらなる開発の効率化を目指して、軽量言語を用いた新しいソフトウェアとハードウェアのコデザイン環境：Mulvery を開発することとした（図 2）。想定ユーザはハードウェアの知識を十分に持たず FPGA 開発に不慣れな開発者とした。そこで、ユーザによる合成のチューニングのフェーズをなるべく最小限とし、ハードウェアの知識がなくともある程度高速なシステムを構築することが可能な仕組みの実現を目指した。

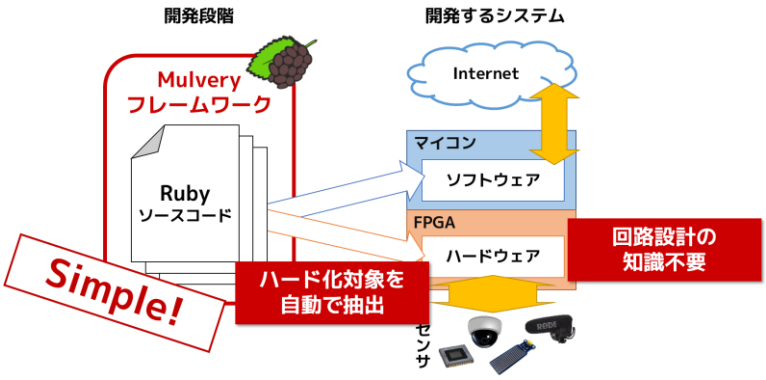

図 2 Ruby のソースコードからハードウェア化の対象を自動的に抽出し回路設計の知識もなるべく必要としない新しいコデザイン環境：Mulvery

3. 開発の内容

本プロジェクトでは CPU+FPGA プラットフォームをターゲットとした、Internet of Things（IoT）のための新しい組込み開発環境を開発した。CPU から FPGA を扱うためのドライバも本開発環境が自動生成するため、ユーザはこれまでのソフトウェア開発と変わらぬ体験で FPGA を導入することができる。

3.1. ソフトウェア/ハードウェア分割機構の開発

ソフトウェア/ハードウェアの分割の戦略として、Reactive Programming のプログラミングパラダイム含むプログラムを合成の対象とした。Mulvery では、Reactive Programming の具体的な実装の一つである Reactive Extensions（以降、Rx とよぶ）を用いて記述された Ruby プログラムをターゲットとした。Rx を用いて記述されたプログラムからは容易にデータフローグラフを抽出することが可能となる性質を持つ。この性質によって、これまでの高位合成系の問題点であった、ハードウェア設計の熟達者によるチューニングのフェーズを省略することを実現した。

本プロジェクトでは、新たなソフトウェア/ハードウェア分割機構の実現手法として、インタフェースクラスを用意することでプログラムの記述をソフトウェア/ハードウェア分割に適した構造に誘導する手法をとった。ユーザは Rx 記述によるデータフローを、インタフェースに既定されたメソッドに実装する。Rx による記述とそうでない記述は互いに疎な結合であるため、異なるメソッドに分離することが非常に容易であるという特徴があるため、ユーザは自然な使用感のまま使用することができる。

3.2. ハードウェアの合成機構の開発

そこで新たなハードウェアを合成する手法として、Rx オペレータと等価なふるまいをするハードウェアをテンプレートとしてあらかじめ記述し、Rx オペレータが呼び出された順にハードウェアを接続していく仕組みを開発した。テンプレートは、図 3 に示すような手法によってハードウェアに合成される。図 3 中に赤字で示した map オペレータは 1 つのモジュールとしてテンプレートからハードウェアが自動生成される。生成されたモジュールはストリームを格納するための S モジュール内にインスタンス化され、前後のモジュールと接続するための配

線が行われる。図 3 中オレンジで示した scan オペレータも同様に、テンプレートからモジュールが合成され、そのインスタンスが map オペレータの直後に接続される。

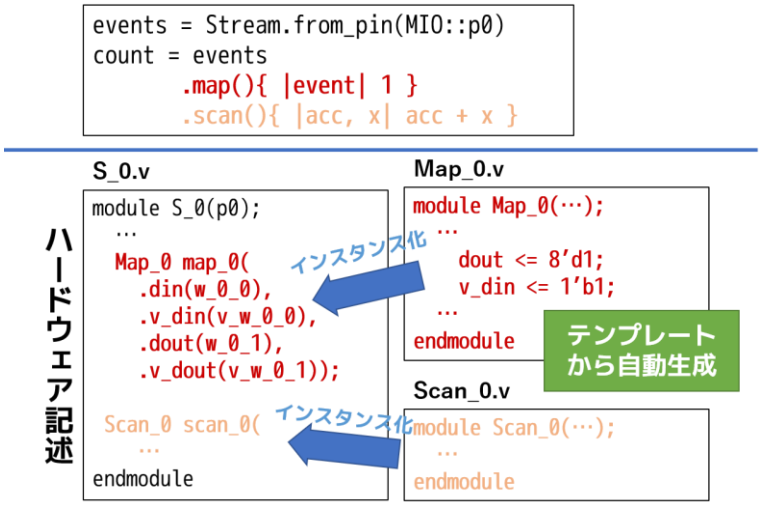

図 3 Rx 記述からハードウェアを合成する手法の概念図

map や scan などは高階関数であるため、ラムダ抽象を受け取ることで具体的な処理の内容が決定する。すなわち、ラムダ抽象の内容もハードウェアに合成する必要がある。ラムダ抽象の内部では Rx 記述を用いないため、テンプレートを用いたハードウェア合成は行うことができない。ラムダ抽象の引数に対してストリームの合成の場合と同様に、マクロのようにふるまうオブジェクトを渡すことでハードウェアの合成を行うものとした。つまり、map オペレータの定義は次のようなものとなる。

```
def map()
  process = yield(HDLComposer.new())
  self.append_module(ERB.new(MAP_HDL).result(binding))
end
```

Ruby 言語では、演算子やメソッドなどをオーバーロードすることができる。これを利用して多段階計算を行う HDLComposer を開発した。

3.3. サンプルアプリケーション

サンプルアプリケーションとして、Worldsemi 社のフルカラーLED WS2812B（以降、NeoPixel とよぶ）を用いた、32×32 の大きさの LED マトリクスを作製した（図 4）。24×1024 ビットの信号を連続して出力することで、すべての NeoPixel を 1 本の信号線で出力することが可能である。

Web サーバの実装には Ruby の Web サーバフレームワークである Sinatra を用いた。これらを動作させるために、独自にビルドした Linux カーネルを用いた Debian OS を用いた。この機構を用いて、C++言語のプログラムから FPGA に画像データを転送し、LED マトリクスに Mulvery ロゴを表示することができた（図 5）。

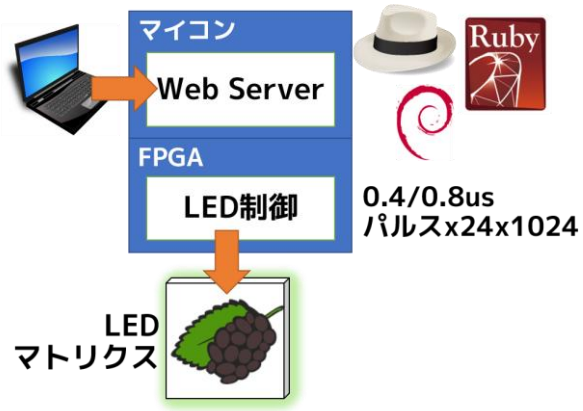

図 4 サンプルアプリケーションの概略図

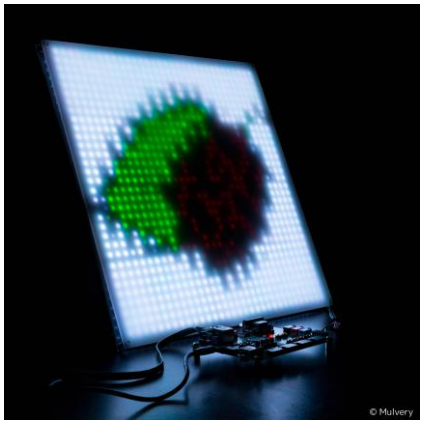

図 5 LED マトリクスに表示した Mulvery ロゴ

4. 従来の技術（または機能）との相違

通常、ソフトウェアを記述するために使用されるプログラミング言語は、逐次実行型の記述である。対してハードウェアを記述するために使用されるハードウェア記述言語は、データフロー型の記述である。この性質の違いから、現在普及している高位合成系やコデザイン環境では、ハードウェア設計に熟達した技術者によるチューニングが必要であった。Mulvery を開発に用いる場合、逐次実行型からデータフロー型への変換が必要なくなるため、チューニングを行わなくとも高いパフォーマンスを発揮するハードウェアを合成することができる。これによって、ハードウェア設計の知識を持たないソフトウェア技術者でも FPGA を利用した IoT デバイスの設計が可能となった。

5. 期待される効果

本プロジェクトでは、Web 開発に広く用いられている動的型付け言語とフレームワークを用いることで、ハードウェアアクセラレーションが実現可能な全く新しい開発環境を実現することができた。近年 IoT の分野は Google IoT Core などのクラウド開発環境も数多く整備され、組込みエンジニア以外のソフトウェア技術者にとっても身近な分野となっている。本システムは Web 関係の技術者にもリーチが可能な、他の高位合成系やコデザイン環境にはない特徴を持ち、より高度な IoT アプリケーション開発のための土壌となることが期待できる。

6. 普及（または活用）の見通し

本システムは Ruby 言語をターゲットとしたフレームワークとして実装をしたが、Ruby 言語に依存しない形に実装を分離、バックエンド化することで、様々なプログラミング言語への移植を容易なものとし、アプリケーションの幅を広げることが可能となる。例えば、Python 言語は人工知能の分野で多く活用されており、Mulvery の移植によって FPGA が人工知能の分野へより活用されやすくなることが考えられる。Rx フレームワークは非常に多くの言語へ移植されているため、それぞれの言語の特徴を生かした活用も可能となる。

7. クリエータ名（所属）

照屋 大地（東京農工大学工学府情報工学専攻）
呉屋 寛裕（富士通クラウドテクノロジーズ株式会社）