C-014

組込み開発向け言語 mruby の FPGA への実装
Implementation of the Programming Language mruby on FPGA

前田 洋征† 田中 和明†
Hiroyuki Maeda Kazuaki Tanaka

1. はじめに

組込みシステムは、IoT (Internet of Things) 化した製品が登場するなど多機能化、高機能化が進んでいる。その結果開発規模が拡大し、開発期間の長期化が問題となっている。そこで組込みソフトウェア開発の効率を向上するために、プログラムが記述しやすく生産性が高い Ruby を組込み開発向けに最適化した mruby が開発された [1]。mruby は、Ruby の特徴を引き継いでおり短い行数でプログラムを作成でき可読性が優れているため組込み開発を効率的に行うことできる。mruby は、Java 言語のように仮想マシンによって実行する仕組みであるため C 言語と比べ実行速度は遅い。

そこで本研究では、組込みシステムにおける一部の機能を、FPGA (Field Programmable Gate Array)を用いてハードウェア化する技術に着目した。FPGA は、内部の論理回路を書き換えることで独自のシステムを構築することができ、同じ処理をソフトウェアで実装した場合よりも高速に処理することができる。近年では、FPGA を用いてハードウェア化することでシステムのオーバヘッドを改善する研究が行われている [2], [3]。本研究では、FPGA 上に mruby の実行環境の構築を行い、実装コストや実行速度の検証を行う。

2. mruby

mruby は、Ruby を軽量化し実行時により少ないメモリ上での動作を可能したオブジェクト指向型スクリプト言語である。従来の Ruby は、Web アプリケーション開発で広く利用され、C 言語と比べてプログラム行数が短く可読性が高い特徴がある。しかし Ruby は、インタプリタ方式であるため実行時に多くのメモリが必要となりメモリ容量に制限がある組込み開発でそのまま利用することは難しい。そこで mruby では、コンパイル方式を採用し専用の仮想マシンによって実行される仕組みになっている。実行の順序を図 1 に示す。

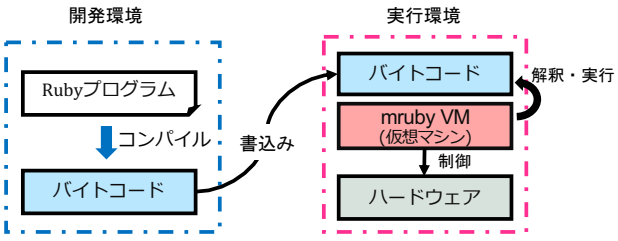

図 1 mruby の実行順序

PC などの開発環境上で作成した Ruby プログラムを専用のコンパイラを用いてバイトコードに変換する。このバイトコードと mruby VM（仮想マシン）をマイコンなどの実行環境に書き込むことで、mruby VM によってバイトコードの解釈と実行が行われる仕組みになっている。mruby は C 言語との親和性が高く設計されており、C 言語から Ruby プログラムを呼び出して利用することが容易にできるようになっている。

2.1 mruby/c

mruby には、小型なデバイスを用いた IoT アプリケーション開発などを想定し、より小さなメモリ上でも動作を可能した mruby/c がある [4]。mruby/c は、従来の mruby から組込み開発に必要な機能に絞り、さらに軽量化されている。従来の mruby とは実行するバイトコードは共通で、仮想マシン (mruby/c VM)が異なっている。従来の mruby は動作するために 400KB のメモリが必要に対して、mruby/c では、50KB 以下のメモリ上で動作することができる。本研究では、mruby/c を対象に FPGA への実装を行う。

3. FPGA への回路構築と mruby/c VM の書き換え

FPGA に mruby/c の実行するための回路を構築し、FPGA の環境に合わせて mruby/c VM の書き換えを行なった。FPGA には、Terasic 社製の FPGA ボードである DE0-Nano を採用した。DE0-Nano は、intel 社製の FPGA である Cyclone IV (EP4CE22)を搭載している。この FPGA は、22,320 個のロジックエレメントと 594 Kbit の内部メモリを使用できる。また、外部メモリとして 32MB の SDRAM を備えている。ボード上には、外部 GPIO ヘッダや LED、プッシュボタン、DIP スイッチを備えている。

DE0-Nano に mruby/c の実行環境を構築するために、intel FPGA に提供されている Nios Ⅱ システムを利用した。Nios Ⅱ は、intel FPGA 内部の論理回路やメモリブロックを使用してソフトウェアコアを構築することができ、周辺回路と組みわせて独自の回路を作成できる。DE0-Nano に構築した Nios Ⅱ プロセッサと周辺回路の構成を図 2 に示す。

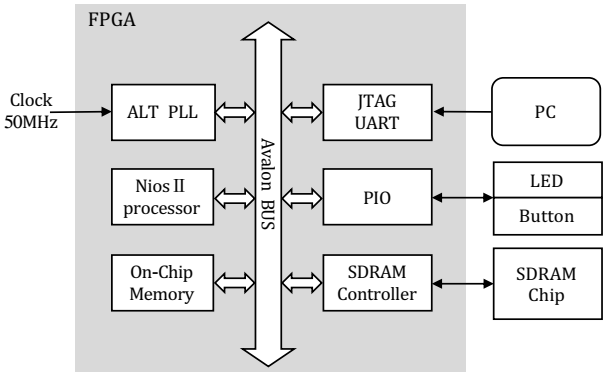

図 2 Nios Ⅱプロセッサと周辺回路の構成

mruby/c VM を書き込むために SDRAM を使用した。メインの記憶領域としては SDRAM を使用するが、高速化のために容量は小さいが処理が速い内部メモリは、実行時のスタック領域として利用する。Nios Ⅱ プロセッサと各モジュールは、専用のバス線である Avalon BUS によって接続する。システムの各モジュールの機能は次の通りである。

- Nios Ⅱ プロセッサ：32bits Nios Ⅱ/e
  (キャッシュ機能が無いプロセッサタイプ)
- On-Chip Memory： FPGA の内部メモリである。

・ALT PLL：ボード上から供給される 50MHz クロックを逓倍する。SDRAM に 100MHz 入力する。
・JTAG UART：PC からのプログラム書き込み用やコンソール出力を行う。
・PIO：外部ピンに繋がった LED やボタンを制御する。
・SDRAM Controller：外部メモリ SDRAM を制御する。

Nios II プロセッサは、C 言語プログラムを実行することができる。そこで、C 言語から Ruby プログラムを呼び出し、mruby/c VM を起動することで実行される形式とした。mruby/c VM は、Nios II の環境に合わせて書き換えを行なった。Nios II と mruby/c には想定するエンディアンに相違があった。Nios II は、リトルエンディアンを想定しているのに対して、mruby/c はビッグエンディアンを想定しているため、Ruby のバイトコードを正常に実行することができない。そこで、ビッグエンディアンからリトルエンディアンに変換するように mruby/c VM の書き換えを行なった。また、mruby/c VM 内のハードウェアに依存する処理をまとめた HAL (Hardware Abstraction Layer)を、Nios II の環境に合わせて記述し、mruby/c VM 内の Nios II 環境上では使用できない機能を別の方法に書き換えて対策を行なった。次に、DE0-Nano の LED やプッシュボタンを Ruby プログラムから制御できるメソッドを mruby/c VM 内に実装した。

Ruby プログラムは、専用コンパイラ を用いて C 言語の配列形式にコンパイルする。実装したメソッドを使った Ruby プログラムとそのコンパイルを図 3 に示す。

```
while true
  button = pio_btn_input
  if button == 0
     pio_led_output 1
  else
     pio_led_output 0
  end
end
```

コンパイル

```
const uint8_t ary[] = {
 0x52,0x49,0x54,0x45,
 0x30,0x30,0x30,0x34,
 0xa8,0xc0,0x00,0x00,
            :
            :
};
```

図 3　Ruby プログラムのコンパイル

mruby/c VM には、ボタンの状態を読み取る pio_btn_input メソッドと、LED を ON/OFF する pio_led_output メソッドを実装した。C 言語からの Ruby プログラムの呼び出しと mruby/c VM を起動するプログラムを図 4 に示す。

```
#include "sys/alt_stdio.h"                    ※一部省略
#include "mrubyc.h"
#include <stdint.h>

int main(void)
    struct VM *vm;
    vm = mrbc_vm_open(NULL);
    if( mrbc_load_mrb(vm, ary) != 0 ) {   ← Rubyプログラム呼び出し
        alt_putstr("Error: Illegal bytecode.");
        return 1;
    }
    mrbc_vm_begin(vm);                    ← mruby/c VM起動
    mrbc_vm_run(vm);
    mrbc_vm_end(vm);
    mrbc_vm_close(vm);
    return 0;
}
```

図 4　C 言語から mruby/c VM の起動

mrbc_load_mrb(vm, ary)によって、ary に入ったバイトコードの配列が読み込まれ、mrbc_vm_begin(vm)によって VM が起動している。図 4 のプログラムを nios2-elf-gcc (Nios II 用 GNU コンパイラ)を用いてビルドを行い、生成した実行ファイルを Nios II システムに書き込むことで、Ruby から正常に LED やプッシュボタンを制御することができた。

4. 性能評価

DE0-Nano に実装した Nios II システムと mruby/c VM の実装コストについて、回路規模と mruby/c のメモリ使用量を表 1 に示す。

表 1 ハードウェア使用量

回路規模 (LEs)	メモリ使用量 (mruby/c VM + Nios II)
3,036 / 22,320 (14%)	95 [KB]

Nios II システムは全体の 14%のロジックエレメント(LEs)を使用しており、比較的小さな規模で構築できた。mruby/c VM には、DE0-Nano 用に LED やプッシュボタンを制御する機能や Nios II のハードウェア情報が追加されるためマイコン等に実装した場合よりも多く、95 KB のメモリを使用する結果となった。次に、実行速度の検証を行った。検証には、整数値 (10～100,00 まで)のカウントアップを繰り返すプログラムを実行し、処理に掛かった時間を測定した。測定には、高精度に時間測定が可能な Nios II の内部タイマを使用した。測定結果を図 5 に示す。

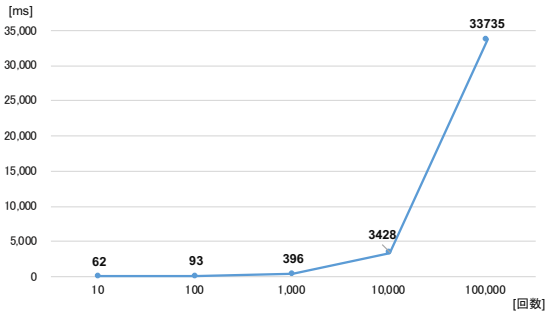

図 5　実行時間

測定の結果、実行速度は遅いという結果になった。この原因として、SDRAM へのアクセスに多くの時間が掛かっていることが考えられる。また、mruby/c VM がバイトコードを読み込む際のビッグエンディアンからリトルエンディアンへの変換が生じることによる遅延が考えられる。

5. 結言

本研究は、mruby/c の実行環境を、Nios II システムを用いて FPGA に実装した。実装の結果、FPGA ボードで正常に mruby/c を実行することができた。今後の課題として、実行速度の向上を行なっていきたい。そのために、外部メモリに SRAM が備わっている FPGA への実装やキャッシュ機能が備わった Nios II プロセッサの利用、エンディアン変換方法の改良を行い、実行速度を改善する。

参考文献

[1] Kazuaki Tanaka, Avinash Dev Nagumanthri, Yukihiro Matsumoto, “mruby – Rapid Software Development for Embedded Systems”, FiSTA 2015 (2015).
[2] 丸山修孝，一場利幸，本田晋也，高田広章，”マルチコア対応 RTOS のハードウェア化による性能向上”，電子情報通信学会論文誌 D, Vol.J96-D,No.10, pp. 2150-2162 (2013).
[3] 田中愛之, 黒柳奨, 岩田彰, “FPGA のためのニューラルネットワークのハード ウェア化手法,” 電子情報通信学会技術報告, NC2000-179, pp.175-182 (2001).
[4] https://github.com/mrubyc/mrubyc.

† 九州工業大学大学院 情報工学府