Vol.2019-OS-147 No.17
2019/7/25

SOTBで実装されたCPUへのmruby/cの移植と評価

大城 研治[1,a)]　濱田　慎亮[1,b)]　並木　美太郎[1,c)]

概要：Internet of Things(IoT) の到来により省電力なプロセッサの需要が増加している．SOTB（Silicon On Thin BOX）を用いて実装されたデバイスでは，超低電圧での動作が可能であり，供給される電圧を変化させることでリーク電流と最大動作周波数のトレードオフを取ることができる．また，組込みシステムにおけるソフトウェアの複雑化に伴い，C 言語では開発コストの増大が問題となっている．これらの課題を解決のため，スクリプト言語を組込みシステムのソフトウェア開発に用いるといった方法が存在する．mruby/c はこの解決策の一つであり，ワンチップマイコンなどの限られたリソース下で動作する Ruby の実行環境である．本研究では、SOTB を用いて実装された CPU である“GC-SOTB”へ mruby/c を移植し評価を行った。

1. はじめに

近年，プロセッサの消費電力の削減は処理性能の向上と並んで最優先課題となっており，Internet of Things(IoT) の到来により省電力なプロセッサの需要が増加している．LSI の省電力には電源電圧を低下させることが効果的である [1]．一方で，電源電圧を下げるために，トランジスタの閾値を下げるとリーク電流が増加する問題がある．また，デバイス素子のばらつき増大による動作下限電圧の悪化により，動作電圧を下げられないため動作電力が増大することも問題視されている．これらの問題を回避する方法として，超低電圧デバイス研究組合（LEAP）が開発した Silicon on Thin BOX（SOTB）技術の使用が挙げられる．SOTB 技術はボディバイアスを用いることで超低電力 LSI を実現することができる．SOTB 技術はマイクロコントローラ，FPGA，アクセラレータなど様々なデバイスに適用され，その有効性が報告されている [2], [3], [4], [5], [6]．ボディバイアス制御を用いた省電力化戦略に関する研究も行われており，[7] では最適な電源電圧とボディバイアスの求め方について述べられている．また, [8] ではリアルタイムシステムにおいての省電力なボディバイアス制御手法について述べられており，34%のエネルギー削減を達成している．また，IoT アプリケーションのような組込みアプリケーションでは低コストかつ高品質が求められている．現状，ほとんどのアプリケーションは C 言語で開発されている．しかし，ソフトウェアの複雑化に伴い，C 言語を用いた開発では多大な開発時間を要するといった問題がある．また，Ruby, Python, JavaScript といったスクリプト言語に比べて学習コストが高いといった問題もある．このような背景から，組込みシステムの開発にスクリプト言語が使用されることがある．mruby は組込みソフトウェア開発に向けて開発された軽量な Ruby である [9]．mruby は Ruby の柔軟性やシンプルさを引き継ぎながらも，組込みシステムにおける限られたリソース下でも動作するように開発されている．よりリソースが限られるワンチップマイコンなどを対象に mruby/c が開発されている [11]．mruby/c は mruby をより軽量にした Ruby の実行環境である．

本研究では，SOTB 技術を用いて実装された CPU である“GC-SOTB”用の評価環境の構築と基礎電力評価，GC-SOTB に mruby/c を移植し実行速度，メモリ消費量について評価を行った．

2. Silion on Thin BOX (SOTB)

Silicon on Thin BOX（SOTB）は，電源電圧を低電圧化させることができ，ボディバイアス制御により，製造後の性能及び電力の両方の最適化を実現することができる．

2.1 ボディバイアス制御

ウェル部の電圧を変更することで閾値電圧を変更することができ，これをボディバイアス制御と呼ぶ．

リバースバイアス時には MOSFET の閾値電圧が上昇し，リーク電流が削減されるが，遅延時間が増大する．フォワードバイアス時は閾値電圧が減少し，遅延時間が減少す

[1] 東京農工大学
Tokyo University of Agriculture and Technology
[a)] oshiro@namikilab.tuat.ac.jp
[b)] Hamada@namikilab.tuat.ac.jp
[c)] namiki@cc.tuat.ac.jp

る代わりにリーク電流が増大する．また，ボディとソースの電圧が等しいとき，これをゼロバイアスと呼ぶ．ボディバイアスを適切に変更することで遅延時間とリーク電流の最適化を行うことができ，高いエネルギー効率を実現することが可能となっている．

2.2 GC-SOTB

GC-SOTB は SOTB を用いて実装された CPU である．MIPS-R3000 アーキテクチャに準拠した Geyser アーキテクチャを採用している．GC-SOTB は大きく分けて core と cache の構成となっている．core には MIPS-R3000 に準拠した CPU や CP0，TLB などが含まれている．また，メモリ転送機能を持つ DMAC（Direct Memory Access Controller）も core に含まれている．cache は，データキャッシュと命令キャッシュから構成されている．core と cache は独立して電源電圧を印加することが可能であり，core 単体での動作も可能となっている．

3. mruby/c

mruby/c は，Ruby の特徴を引き継ぎつつ，プログラム実行時に必要なメモリ消費量が従来の mruby より少ない Ruby の実行環境である．図 1 に従来の組込みシステムと mruby/c を用いたシステムを示す．mruby/c は OS を使用せずに Ruby プログラムを実行することができる．また，メモリ消費量が小さく，限られた資源や電源環境下においての運用に適している．mruby/c では，バイトコードを実行する仮想マシン（mruby/c VM）がハードウェアに依存した特徴を吸収してくれるため，Ruby でソフトウェア開発を行い C 言語でハードウェア開発を行うといったようにソフトウェア開発とハードウェア開発を切り離すことができる．そのため，ハードウェアが異なっても mruby/c VM を変更するだけでソフトウェアを移植することができる．サーバではスクリプト言語を用いて情報システムを開発されるので，IoT のエッジ側でもスクリプト言語により，プログラムを開発することでサーバ側と処理の連携を行いやすくなるといった利点もある．図 2 に mruby/c におけるアプリケーション開発の概要を示す．mruby/c を用いたアプリケーション開発では，PC 上で開発した Ruby プログラムを mruby コンパイラを使用しコンパイルしバイトコードを得る．このバイトコードをターゲットデバイス上の mruby/c VM で実行するといった形で開発を行っていける．

4. 評価環境

SOTB チップでは，一般的なデバイスとは異なり電源電圧に加えボディバイアス電圧を供給する必要があり，また電源電圧，ボディバイアス電圧，動作周波数を変化させることで基盤特性を変更できるといった特徴からこれらの要

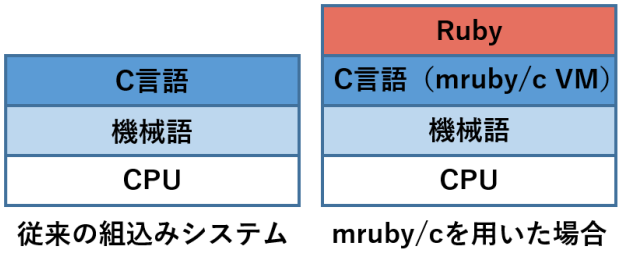

図 1 従来の組込みシステムと mruby/c

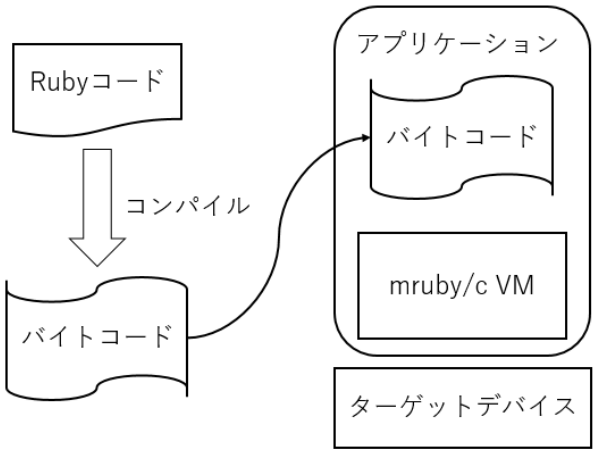

図 2 mruby/c によるアプリケーション開発

素を変更することのできる環境が必要となる．

本研究において実装した評価環境では，SOTB チップのボディバイアス制御に必要な，電源電圧，ボディバイアス電圧，動作周波数を SOTB チップ上で動作する OS/アプリケーションから制御することができる．図 3 に評価環境の構成を示す．SOTB チップの電源電圧（VDD）およびボディバイアス電圧は，Power Supply Unit (PSU) によって供給される．MicroZed が提供するリソースは GC-SOTB とバスコントローラで接続されており，GC-SOTB から利用することができる．PSU は SOTB チップから I2C を経由して制御することができ，供給する電圧を動的に変更することができる．同様に MicroZed が提供する PLL はリコンフィグレーション可能となっており，動的に SOTB チップに供給するクロックの周波数を変更することができる．OS/アプリケーションを動作させるために必要な RAM，ROM などの周辺機能は MicroZed が提供する．SOTB チップの評価に必要な電流データ，電圧データは I2C で接続された電流・電圧計から取得することができる．

5. GC-SOTB に移植するための課題

GC-SOTB に移植するためには以下の課題がある．

1. Ruby からの I/O 制御の実現
2. Ruby で割り込みを扱う機能の実現

SOTB 技術では，電源電圧とボディバイアス電圧を変更

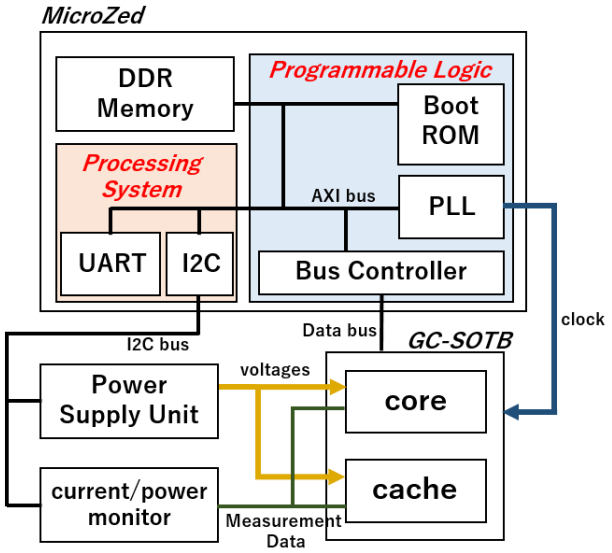

図 3　評価環境のハードウェア構成

することで消費電力を削減することができる．CPU の消費電力は，CPU の使用率やキャッシュメモリの使用率などに作用される．つまり，実行するプログラムによって消費電力は異なる．また，一つのプログラムでも常に一定の動作をするわけではないので，プログラム実行中にも消費電力は変化する．プログラムの動作に合わせて電圧，動作周波数を変化させることができればより効果的な消費電力の削減が期待できる．そのため，mruby/c を GC-SOTB に移植する際に，電圧，動作周波数を変更する機能を追加する．GC-SOTB の評価環境では，電圧は PSU が，動作周波数は MicroZed 内の PLL が提供している．つまり，これらのデバイスを制御する，I/O 制御の機能が必要ようとなる．

2 つ目に，省電力化を行っていく上で割り込みの機能が必要となってくる．mruby/c では割り込み対するサポートが存在しない．そのため，割り込みを Ruby で処理することができない．割り込みは，GC-SOTB がスリープ状態から復帰する際の契機となる．また，外部デバイスからの割り込みを処理するためにも，割り込みへのサポートは必要となる．

6. mruby/c の GC-SOTB への移植

mruby/c を GC-SOTB に移植した．mruby/c のソースコードは [11] にて公開されている．本研究では，beta4 版を使用している．本評価環境では GC-SOTB が利用する I/O は MicroZed が提供している．この MicroZed 内のデバッグ用シリアル出力に用いる UART や，PSU を制御するための I2C のデバイスドライバを実装した．GC-SOTB は，チップ内部にサイクルカウンタやタイマ，キャッシュのパフォーマンスカウンタなどの I/O レジスタを備えている．mruby/c の性能評価の際に GC-SOTB の実行サイクルを取得する必要があるため，これらのレジスタからデータを取得するための実装も行った．また，Ruby コードから

表 1　追加したメソッド

メソッド名	詳細
set_clock(clk)	動作周波数を clk に変更 (MHz 単位)
set_voltage(ch, val)	チャネル ch の電圧を val に設定 (mV 単位)
get_current()	現在の GC-SOTB の電流を取得
get_voltage()	現在の GC-SOTB の電圧を取得
read8(address)	address の I/O ポートから 8 ビット読み取る
read16(address)	address の I/O ポートから 16 ビット読み取る
read32(address	address の I/O ポートから 32 ビット読み取る
write8(val, address)	address の I/O ポートへ val を 8 ビット書き込む
write16(val, address)	address の I/O ポートへ val を 16 ビット書き込む
write32(val, address)	address の I/O ポートへ val を 32 ビット書き込む
mask	割り込みを禁止する
unmask	割り込みを可能にする
set_excep(excep , method)	割り込み excep に対してメソッドを登録する

I/O 制御を行うためメソッドを追加する（表 1）．mruby/c では，コンフィグファイルによりファイル I/O や浮動小数点を使用するか設定できるようになっている．GC-SOTB には FPU が存在しないため浮動小数を使用しないように設定している．

7. mruby/c と電源制御

図 4 に電源制御機構の概要図を示す．mruby/c から電圧や動作周波数を変更するために，図 4 における，Power Supply Units（PSU）および，PLL を制御する必要がある．PSU については，mruby/c を移植する際に作成した I2C のデバイスドライバによって制御することができる．PLL についてはバスコントローラによって MicroZed 内のリソースにアクセスできるようになっているおり，GC-SOTB からはメモリマップド I/O として制御することができる．mruby/c の機能を用いて制御用のメソッドを提供することで，Ruby からも電源制御機構を利用することができる．電源制御機構と mruby/c によって Ruby から GC-SOTB の電源電圧・動作周波数を動的に変更することができる．

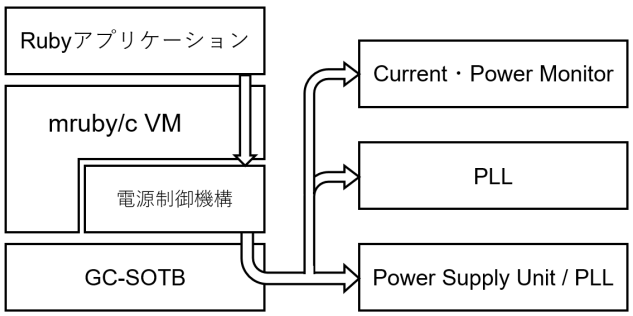

図 4　電源制御機構

7.1 mruby/c と割り込み

mruby/c での割り込み処理を図 5 に示す．VM は opcode を実行する流れの中で，opcode 実行後に割り込みフラグ

が立っているか確認する．割り込みがあった場合，割り込みに対応した処理を実行する．具体的には，割り込みに対応したコールバックメソッドを呼び出す．コールバックメソッド呼び出しは，VM の opcode である OP_SEND を呼び出すことで実現できる．

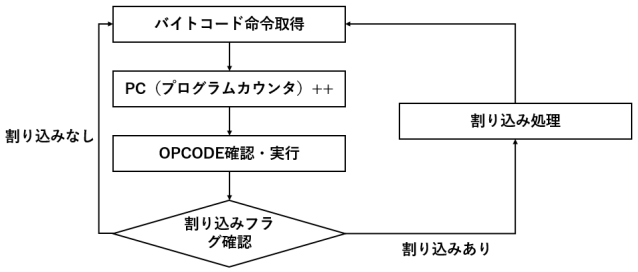

図 5　mrubyc/ct と割り込み処理

8. mruby/c の性能評価

GC-SOTB に移植した mruby/c を実行速度，メモリ消費量において評価した．

8.1 実行速度

GC-SOTB の比較対象として PSoC　5LP Prototyping Kit に mruby/c を移植し実行速度の比較を行った．要素 10 個の配列のバブルソートを行ったときの実行速度を計測する．この時，C 言語で直接記述したものと，Ruby で記述し mruby/c 上で実行したものをそれぞれ計測する（図 7,8）．GC-SOTB においては，プログラムデータは図 3 における，左上の DDR Memory に配置している．表 2 にバブルソート実行時の実行サイクル数を示す．PSoC 5LP は 24MHz，GC-SOTB は 25MHz で動作している．表 2 の GC-SOTB のキャッシュ無しと PSoC を比較すると，GC-SOTB での実行サイクル数が PSoC に比べて，C でのソートでは約 6 倍なのに対して，Ruby では約 22 倍と同じ比率とならなかった．mruby/c VM がバブルソートと比較してメモリバウンドなプログラムであるため，キャッシュのある PSoC においてより高速化がなされたものと考えらえる．次に，GC-SOTB のキャッシュ有とキャッシュ無しの比較を行っていく．C で記述されたバブルソートについては，キャッシュ無しの場合，キャッシュ有に比べて 15 倍，Ruby では 12 倍の実行サイクル数を要している．mruby/c 上で実行した Ruby に比べて C の実行サイクルがよりり削減できたことについては，C のバブルソートの方がよりキャッシュによる高速化の効果が表れたものと考えられる．C でのバブルソートはテキスト，データの量が少なくキャッシュミスの発生率が低くなり，mruby/c では Ruby コードの実行時にもバイトコードのみでなく，オブジェクト，各バイトコードに対応した関数の呼び出しなどより広い範囲のアドレスにアクセスすることになるためキャッシュミスが C と比べて多くなったものと考えられる．

最後に，PSoC 5LP と GC-SOTB（キャッシュ有）について比較していく．C で記述されたバブルソートについては，PSoC の方が実行サイクル数が 2.4 倍多くなった．一方で Ruby については，GC-SOTB の方が 1.8 倍多くなった．C では GC-SOTB の方が実行サイクル数が少ないのに，Ruby の場合は PSoC の方が実行サイクル数が少なくなった．上記での述べたように，GC-SOTB で C のバブルソートを実行した際，テキスト，データのサイズが小さいことからキャッシュによる高速化の効果が顕著に表れたものと考えられる．一方で Ruby ではキャッシュによる高速化はあったものの C の場合ほどではなかったものとみられる．それに加えて，本研究で用いている図 3 に示す GC-SOTB の評価環境では，プログラムデータを配置している DDR Memory へのアクセスに，バスコントローラでのアドレス変換と AXI バスを経由することからオーバヘッドが比較的大きいものとなっている．これもメモリアクセスが C の場合よりも多い Ruby での実行速度が C ほど高速でなかった要因の一つと考えられる．

表 2　バブルソートの実行サイクル数

	C	Ruby
PSoC 5LP	1132	234272
GC-SOTB（キャッシュ有）	464	431250
GC-SOTB（キャッシュ無）	6975	5103769

8.2 メモリ消費量

速度の次に mruby/c のメモリ消費量について評価した．7.1 章にて用いたバブルソートを実行するプログラムの静的なメモリ消費量を表 3 に示す．今回使用したプログラムでは mruby/c VM 本体だけでなく，GC-SOTB に移植する際に追加したコードも含まれている．また，VM がオブジェクトを生成した際に消費されるメモリ領域は，コンパイルの段階であらかじめ静的に確保されており，オブジェクトを生成するとそのメモリ領域から割り当てられる形になっている．この領域は.bss セクションに存在し，初期状態では 2560 バイト確保されている．実行速度評価に用いたプログラムでは 1172 バイト消費されていた．

次に Ruby で整数の配列を確保した際にどれだけのメモリが使用されるかについて調査を行った．表 4 に配列のサイズと使用されたメモリ量の関係を示す．表 3 の結果から，配列のサイズが 2 の時を除いて 16Byte ずつ増加していることが読み取れる．サイズ 1 の時が 36Byte なことから，36Byte のうち 20Byte がヘッダ，16Byte が配列のデータとなっていると考えられる．

9. GC-SOTB の基礎電力評価

実装した評価環境において，MiBench の bitcount を用

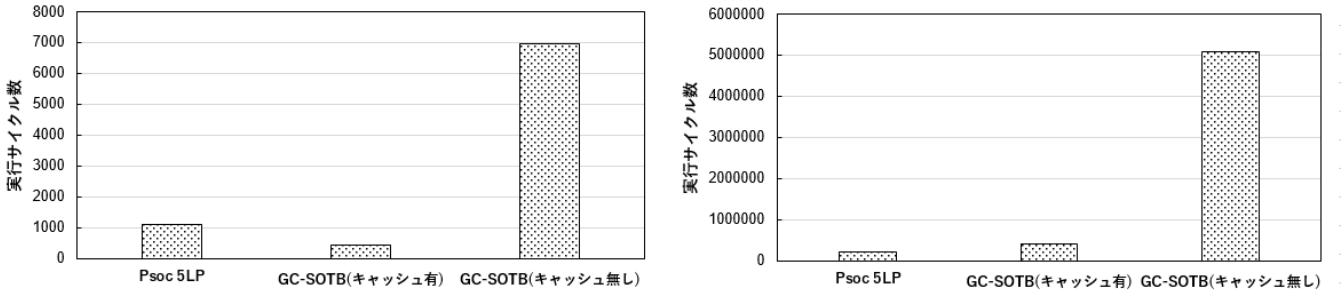

図 6 バブルソートの実行サイクル（左図:C, 右図:Ruby）

```
int i,j,tmp;
int array[] = {4,5,1,2,9,8,3,6,7,10};
for(i = 0; i < 9; i++){
  for(j = 9; j > i; j--){
    if(array[j-1] > array[j]){
      tmp = array[j-1];
      array[j-1] = array[j];
      array[j] = tmp;
    }
  }
}
```

図 7 バブルソートの C コード

```
def sort()
  ary = [4,5,1,2,9,8,3,6,7,10]
  i = 0
  while i < 9 do
    j = 9
    while j > i do
      if ary[j-1]>ary[j] then
        tmp = ary[j-1]
        ary[j-1] = ary[j]
        ary[j] = tmp
      end
    j = j - 1
    end
  i = i + 1
  end
end
```

図 8 バブルソートの Ruby コード

表 3 静的なメモリ消費量

セクション	サイズ（Byte）
.text	39130
.rodata	2734
.sdata	4
.bss	48
.sbss	36932
合計	78848

いて GC-SOTB の電力評価を行った．評価結果を表 1 に示す．電流値は bitcount 実行中に 100ms 間隔で計測したものの平均となっている．表 1 の消費エネルギーは，bitcount

表 4 整数配列のメモリ消費量

配列のサイズ	メモリ消費量（Byte）
1	36
2	52
3	68
4	84
5	100
10	180

を実行するのに必要な消費エネルギーとなっている．
表 1 の結果から，電源電圧を低下させることで，消費エネルギーが減少していることが確認できる．VDD=0.7V 時の消費エネルギーと比較すると，VDD=0.6V では 25.3%, VDD=0.5V では　46.7%の消費エネルギー削減となっている．一方で，同じ VDD において動作周波数を変更したときは，電流値は減少しているが，消費エネルギーで見ると変化はないことが確認できる．動作周波数を低下させると消費電力は減少するが実行時間が増加するので，消費エネルギーに変化がなかったと考えられる．VDD=0.6V, 15MHz，VBP=0.7V, VBN=-0.1V のリバースバイアスをかけた時は，ゼロバイアスと比較して，4.2 %の消費エネルギーの削減となっている．VBP=0.8V, VBN=-0.2V として，より強いリバースバイアスかけてもそれ以上の消費エネルギー削減とはならなかった．次に，VDD=0.5V, 15MHz, VBP=0.0V, VBN=0.5V のフォワードバイアスをかけた時は，VDD=0.5V のゼロバイアス時と比べ消費エネルギーが 73.1%増加している，同じ周波数である VDD=0.6V, 15MHz のゼロバイアスと比べ 23.3%増加している．

以上の結果から，VDD を低下させることが最も消費エネルギーを削減するのに効果的であり，リバースバイアスを併用することでより多くの消費エネルギーを削減できることが確認できた．一方で，フォワードバイアスは最大動作周波数を引き上げる効果があるが，VDD を上げて最大動作周波数を上げる方がエネルギー効率が良くなることが確認できた．

SOTB において消費エネルギーを削減するには，VDD を変更することが重要となる．リバースバイアスを用いることでリーク電流を削減できるため消費エネルギーを削減することが可能であるが，VDD を変更するよりも影響が

少ない．しかし，スリープ状態ではリーク電流によって消費されるエネルギーが比較的多くなるため，リバースバイアスの影響が大きくなってくる．一方，フォワードバイアスは，VDD を上げて最大動作周波数を上げる方がエネルギー効率が良くなるため使用する機会は少ない．供給できる電圧が限られており VDD を上限まで上げても目的の周波数に届かないといった状況において，フォワードバイアスを併用することで周波数をより引き上げるといった使用方法が考えられる．

表 5 bitcount 実行時の電流値と消費エネルギー

VDD (V)	電流値 (mA)		消費エネルギー (mJ)		
	core	cache	core	cache	core+cache
0.7 (30MHz)	7.1	33.9	5.6	26.5	32.1
0.7 (25MHz)	6.0	27.9	5.6	26.5	32.1
0.6 (20MHz)	4.1	19.6	4.2	19.8	24.0
0.6 (15MHz)	3.1	14.7	4.3	19.7	24.0
0.5 (10MHz)	1.7	8.4	3.0	14.1	17.1
*0.6 (15MHz)	2.9	14.4	3.8	19.2	23.0
**0.6 (15MHz)	2.9	14.3	3.8	19.2	23.0
***0.5 (15MHz)	5.5	20.7	6.3	23.3	29.6

*VBP=0.7V, VBN=-0.1V
**VBP=0.8V, VBN=-0.2V
***VBP=0.0V, VBN= 0.5V

10. おわりに

本稿では，SOTB で実装された CPU である GC-SOTB の基礎電力評価および mruby/c の移植，性能評価を行った．

mruby/c の性能評価では，キャッシュを有効にした GC-SOTB において，C 言語に比べて Ruby では 929 倍の実行サイクルとなった．また，PSoC と GC-SOTB のキャッシュ無しを比較した際に，キャッシュによる高速化の恩恵が大きかったことから mruby/c VM は比較的メモリバウンドなプログラムであると推測される．メモリ消費量においては，実行速度評価に用いたバブルソートのプログラムサイズが 77KB であった．mruby/c では静的に確保されたメモリ領域をオブジェクトに割り当てるようになっている，バブルソートのプログラムでは 1172 バイト消費されていた．電力評価の結果から，消費エネルギーを削減するためには電源電圧を低下させることが最も効果的であり，リバースバイアスを併用することでより多くの消費エネルギーを削減できた．一方で，フォワードバイアスでは最大動作周波数を引き上げる効果があるものの，エネルギー効率が悪化するといった結果が得られた．今後の課題として，mruby/c のより詳細な性能評価と電力制御機構と割り込み処理の実装が挙げられる．本稿で行った性能評価では，小規模なバブルソートでのみの評価であった．今後，様々なプログラムを実行した際の性能を評価していきたいと考えている．

11. 謝辞

本研究は，JSPS 科研費　基盤研究 (B) ビルディングブロック型計算システムにおけるチップブリッジを用いた積層方式（18H03215）に関する研究の助成を受けた．

参考文献

[1] Ronald G. Dreslinski, et. al.“ Reclaiming Moore's Law Through Energy Efficient Integrated Circuits”, Proceedings of the IEEE, Vol.98, No.2 pp.253-266 Feb. 2010.
[2] Takashi Ishigaki, et al. “Ultralow-power LSI Technology with Silicon on Thin Buried Oxide (SOTB) CMOSFET”, Solid State Circuits Technologies, INTECH, pp.145-156, 2010
[3] N. Sugii, el al. “Ultralow-Power SOTB CMOS Technology Operating Down to 0.4 V”, Journal of Low Power Electronics and Applications. 4, pp.65-76, Appl. 2014.
[4] H. Su, et al. ”Body bias control for a coarse grained reconfigurable accelerator implemented with Silicon on Thin BOX technology,” 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, pp. 1-6, 2014.
[5] H. Masakazu, et al. “SOTB Implementation of a Field Programmable Gate Array with Fine-Grained Vt Programmability”, Journal of Low Power Electronics and Applications, Vol. 4, No.3 pp. 188-200, 2014
[6] K. Masuyama, et al.“A 297mops/0.4mw ultra low power coarse-grained reconfigurable accelerator CMA-SOTB-2”, 2015 International Conference on ReConFigurable Computing and FPGAs (ReConFig), pp. 1-6, 2015
[7] H. Okuhara, et. al. “An optimal power supply and body bias voltage for a ultra low power micro-controller with silicon on thin box MOSFET”, IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED), July 2015
[8] CCC. torres, et. al.“Analysis of Body Bias Control Using Overhead Conditions for Real Time Systems: A Practical Approach”, IEICE Transactions on Information and Systems, Vol.E101.D, No.4, pp.1116-1125, 2018
[9] K. Tanaka, et al.“mruby – Rapid Software Development for Embedded Systems”, 2015 15th International Conference on Computational Science and Its Applications, pp. 27-32, 2015
[10] T. Yamamoto, et al. “Component-Based Framework of Lightweight Ruby for Efficient Embedded Software Development”, Journal of Computer Software, Vol. 34, No. 4, pp. 3-16, 2017
[11] mruby/c, https://github.com/mrubyc/mrubyc
[12] 森脇淑也, 田中和明, “組み込み向け Ruby の開発と実装に関する研究: RiteVM の実装と検証について”，電気関係学会九州支部連合大会講演論文集，Vol. 2012, pp. 403-403, 2012