

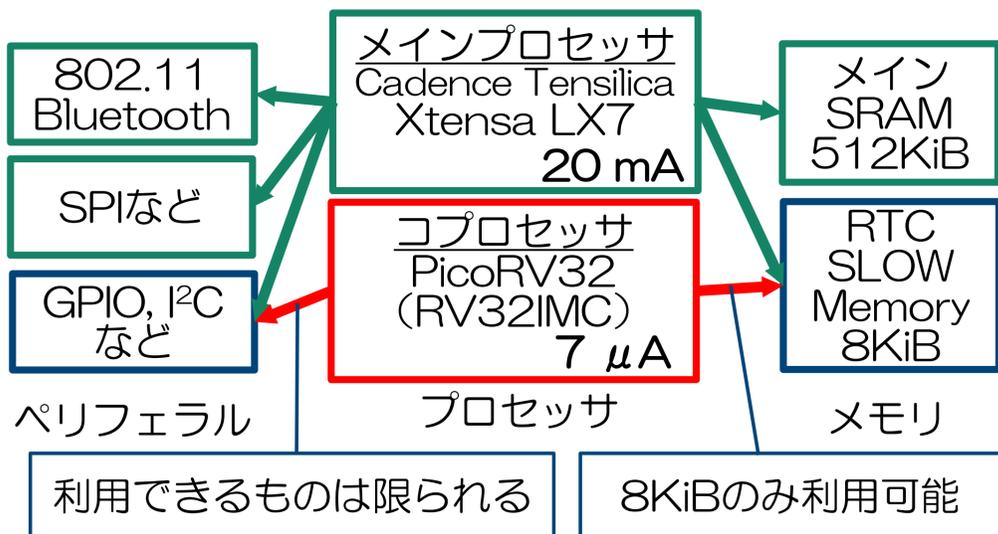
マイコンの省電力コプロセッサを活用できる動的コンパイラの実装に向けて

鈴木豪, 渡部卓雄, 森口草介 (東京工業大学情報理工学院)

ターゲット: Espressif ESP32-S3

メインプロセッサの他に, RV32IMCプロセッサ (RISC-V) である省電力コプロセッサ (Ultra Low Power Coprocessor) を持つ。

ESP32-S3のシステム概要図



典型的消費電流は公式データシート[1]の最小値による

コプロセッサプログラミングの難しさ

- メモリの一部領域しか使えない。
- C言語しか使えない。(機械語のみのマイコンもある。)
- メモリマッピングが異なるマイコンもある。(ESP32-S3はこれに該当)

mruby言語とmruby/c^[2] インタプリタ

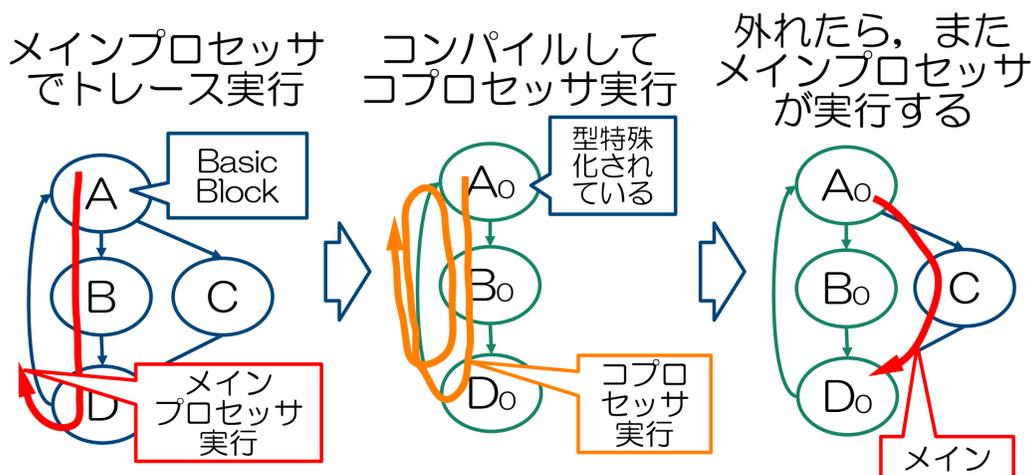
Rubyをmrubyバイトコードに変換してから実行する。ESP32-S3のメインプロセッサにおけるフットプリントは50KiB前後であり, コプロセッサのメモリに載せるのは厳しい。本研究ではmruby/cを拡張する。

mruby言語仕様とmruby/c処理系の特徴

- break/nextは大域脱出が必要, 例外, クロージャ
- オブジェクトは固定長 (mruby/cは動的にフィールドを増やす操作に対応していないため, Static TypeScript[3]と似ている。)

実装方針

メインプロセッサがコプロセッサ向けのプログラムを実行時コンパイルする。

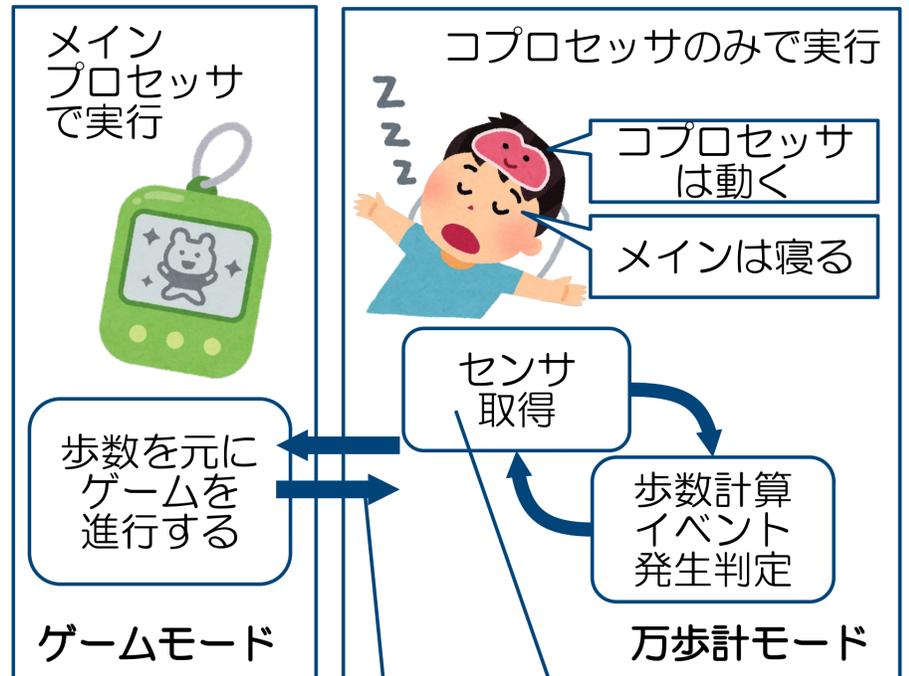


関連手法

- Tracing JIT[4]: 翻訳単位がBasic Blockではないため, 分岐によってコードサイズが大きくなる。
- Lazy Basic Block Versioning [5]: GenericなVersionを生成するため, コードサイズが大きくなる。

省電力コプロセッサのユースケース

ゲーム要素のある万歩計



本研究での実装例

```
steps = Copro.sleep_and_run do
  sum_steps = 0
  while !button.push? do
    sum_steps += MotionSense.get()
    # イベントチェックなど ...
  end
  sum_steps
end
if イベント?(steps) then
  # LCDを光らせたり...ネットワーク通信したり...
end
```

このブロックはコプロセッサで実行される

(真面目な実用例: 農業用のIoTセンサなど)

チャレンジと今後の予定

コード生成

コンパイルと転送をするプログラムの範囲の決定

- 型特殊化したコードはコードサイズをどの程度減らすことができるかを調べる。
- どの程度頻度の低いコードパスをメインプロセッサで実行させるべきかを考える。
- メインプロセッサを起こすオーバーヘッドに気を付ける必要がある。

メモリ管理

利用可能なメモリ空間は8KiB (ランタイムのコードも含む必要がある。)

- 断片化への対処とコードの配置方法

[1] Espressif Systems. ESP32-S3 Series Datasheet v1.8, 2023

[2] しまねソフト研究開発センター, 九州工業大学. mruby/c <https://github.com/mruby/mruby>

[3] Thomas Ball, Peli de Halleux, and Michal Moskal. 2019. Static TypeScript: an implementation of a static compiler for the TypeScript language. (MPLR 2019). DOI: 10.1145/3357390.3361032

[4] Andreas Gal, Christian W. Probst, and Michael Franz. HotpathVM: an effective JIT compiler for resource-constrained devices. (VEE '06). DOI: 10.1145/1134760.1134780

[5] Maxime Chevalier-Boisvert and Marc Feeley. Simple and Effective Type Check Removal through Lazy Basic Block Versioning. (ECOOP 2015). DOI: 10.4230/LIPIcs.ECOOP.2015.101