TaC シミュレータマニュアル Ver. 0.1.0

徳山工業高等専門学校
情報電子工学科

目次

第 1 章

TaC シミュレータの概要

TaC シミュレータは, 徳山高専で開発されている教育用 16bit コンピュータ TaC をパソコン上で再現するソフトウェアである. 本来であれば, TaC を使うためには実機が必要だが, シミュレータを使うことで実機が無くても TaC の演習を行うことができる. また, TypeScript 言語で記述されており, VHDL で記述された実機よりも機器や機構の追加が容易である.

1.1 目標

TaC の動作再現 TaC シミュレータは, 最終的に授業の演習や TacOS の開発などを手助けするツールとして利用できるようにしたい. そのため, 実機との動作の違いなどを無くすことを目標とする.

拡張容易性 TaC 実機は VHDL で記述されており, ハードウェア化するためにタイミングや回路規模等の制約が多く, 新しい機能を追加するためのコストが大きい. 一方で, ソフトウェアで実現される TaC シミュレータは改造が容易なので新しい機能の検証を手軽に行うことができる.

デバッグ用機能 TaC の検証のために, デバッグ用の機能を追加することもできる. 例えば, 特定のプロセスを実行している間に何回命令が実行されたかを計測する機能など, 実機に搭載するのが難しい機能についても, シミュレータならば実現可能である.

ブラウザ上での動作 TaC シミュレータは TypeScript 言語で記述されているが, これは将来的にブラウザ上で TaC を利用できるようにするためである. これにより, TaC を所有していない人が TaC がどのように動作するのかを気軽に知ることができる. また, 実機が故障した場合の代替手段としても利用できる.

1.2 開発環境

2023 年 1 月現在の TaC シミュレータは, Electron というソフトウェアフレームワークを使用して開発されている. これにより, Web アプリケーションの開発と同じように HTML, CSS, JavaScript を使用してスタンドアロンアプリを開発することができる. また, 使用言語には JavaScript の静的な型付けスーパーセットである TypeScript を使用している.

1.3 開発記録

令和 2 年度卒業研究 JavaScript を用いて TaC のシミュレータの開発を開始．CPU とメモリ部分の実装が終了

令和 3 年度卒業研究 Electron を導入し, スタンドアロンアプリとして TaC のシミュレータの開発を行う．割込みコントローラや I/O 機器などの実装が完了し, IPL プログラムが動作するようになった.

令和 4 年度卒業研究 使用言語を TypeScript に変更．MMU の実装やその他バグの修正, 設計の見直しを行い, 仮想記憶を使用して動作する TacOS が起動し動作するようになった.

1.4 実機との違い

TaC シミュレータには TaC 実機といくつか違う仕様が存在する

- TaC 実機とブレークポイントの挙動が異なる．実機では CPU が指定したアドレスにアクセスしたときに停止するが, シミュレータでは PC が指定したアドレスになったときに停止する．また, ブレークする番地はシミュレータの左下にあるテキストボックスから入力するため, 実機のように MA レジスタの値をブレークポイントとして使用することはできない.
- 実機は RESET ボタンを押すと自動的に実行されるが, シミュレータでは RUN ボタンを押す必要がある.
- 実機では, リセット時に SETA ボタンが押されていた場合に”\kernel.bin” ファイルの代わりに”\kernel0.bin” ファイルを読み込む機能があるが, シミュレータには無い.
- TaC のタイマーは 1ms 毎にカウントを進めるが, シミュレータでは動作環境の都合上遅延が発生するため, 正確に 1ms 毎にカウントが進むわけではない.
- 実機と I/O マップが異なる．詳しくは付録 B を参考のこと.

第 2 章

インストール方法

2.1 Node.js のインストール

TaC シミュレータのビルドには Node.js 環境が必要になるため, まずは Node.js のインストールを行う. https://nodejs.org/ja/ に移動してインストーラをダウンロードし実行することでインストールできる. 正しく実行できたかどうかを確認するためには, 以下のコマンドを実行する.

```
$ node --version
v16.15.0
```

2.2 TaC シミュレータのインストール

TaC シミュレータのソースコードは https://github.com/TacSimTeam/TacSimulator-TS から入手できる. ダウンロードした配布物を展開し以下の順にインストールを行う.

2.2.1 依存パッケージのダウンロード

TacSimulator-TS ディレクトリに移動し以下のコマンドを実行する.

```
$ npm ci
```

以上で TaC シミュレータのビルドに必要なパッケージを全てインストールできる.

2.2.2 TaC シミュレータのコンパイル

以下のコマンドを実行するとソースコードがコンパイルされる.

```
$ npm run compile
```

2.3 TaC シミュレータの実行

TacSimulator-TS ディレクトリで以下のコマンドを実行すると, TaC シミュレータが起動する.

```
$ npm start
```

成功すると, 画面に TaC シミュレータのウィンドウが表示される.

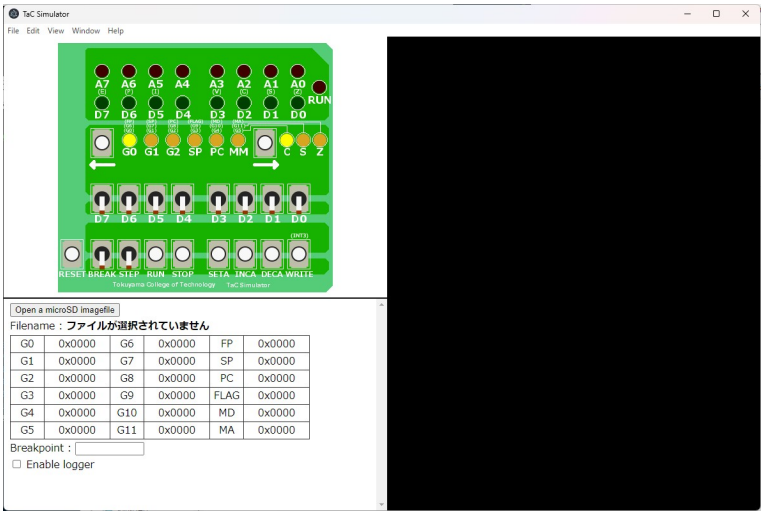

図 2.1 TaC シミュレータの画面

2.4 TaC シミュレータの実行バイナリ作成

2.4.1 Windows の場合

以下のコマンドを実行することで, TaC シミュレータのインストーラを生成することができる.

```
> npm run build:win
```

成功すると, TacSimulator-TS/TacSimulator に TacSimulator Setup X.X.X.exe というファイルが生成され, 実行すると TaC シミュレータのインストーラが実行される.

PC にインストールしたくない場合は, TacSimulator-TS/TacSimulator/win-unpacked に実行バイナリが生成されているので, これを実行することでシミュレータを起動できる.

2.4.2 macOS の場合

以下のコマンドを実行することで, TaC シミュレータのインストーラが入った dmg ファイルを生成することができる.

```
> npm run build:mac
```

成功すると, TacSimulator-TS/TacSimulator に TacSimulator Setup X.X.X.dmg というイメージ

ファイルが生成される. 中にはインストーラが入っており, 実行するとインストールが開始される.

PC にインストールしたくない場合は, TacSimulator-TS/TacSimulator/mac に実行バイナリが生成されているので, これを実行することでシミュレータを起動できる.

第 3 章

操作方法

3.1 TaC シミュレータの画面

図 3.1 に TaC シミュレータの画面を示す．①には TaC のコンソールが描画されており，実機と同じようにスイッチとボタンを使用することができる (実機の操作方法は https://github.com/tctsigemura/TeC7/blob/master/Doc/Manual/manual.pdf を参考のこと)．②にはマイクロ SD の代わりに使用するディスクイメージの読み込みボタンやレジスタの内容出力，ブレークポイントの設定といったデバッグ用の機能が表示されている．③は TaC シミュレータのターミナルである．

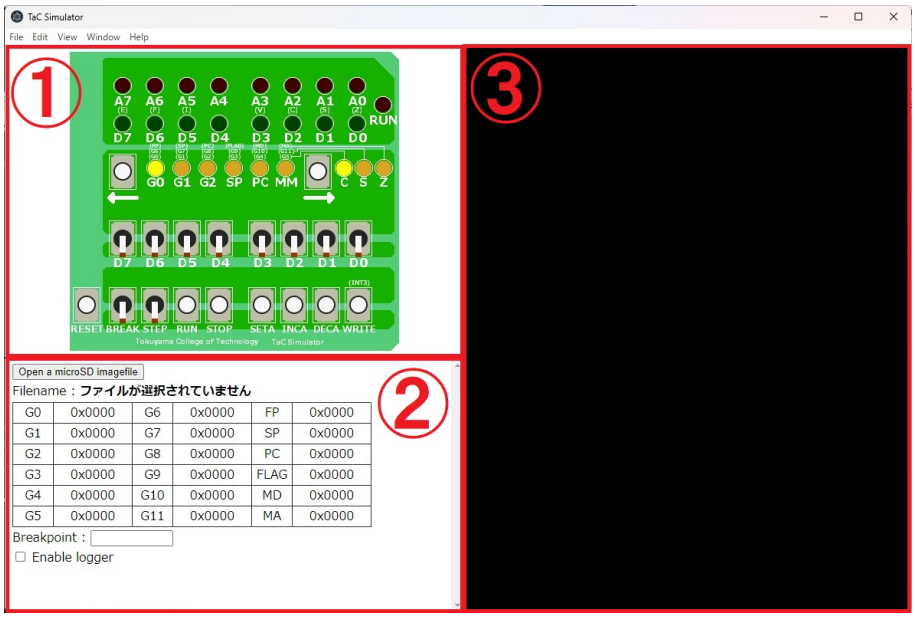

図 3.1 TaC シミュレータの画面

3.2 イメージファイルの実行

コンソールの RUN ボタンをクリックすることで，シミュレータに予め書き込まれている IPL プログラムが起動する．ただし，初期状態ではマイクロ SD が読み込まれていないため，ターミナルに次のよう

な表示を出力して動作を停止する.

```
TaC Initial Program Loader Ver.4.5.0(TeC7d v2.0.0)
(build date : Fri Oct 7 15:16:36 JST 2022)
Copyright(c) 2009-2022 Tokuyama College of Technology
All rights reserved.

Please insert the card.
Push "RESET" to boot the kernel
```

TaC シミュレータではマイクロ SD の代わりに FAT16 ファイルシステムでフォーマットされたディスクイメージファイルを使用できる (以後, マイクロ SD イメージファイルと呼ぶ). TacSimulator-TS/sample ディレクトリに入っているサンプル用のマイクロ SD イメージファイルを使用することができる. また, 自分で作成することもできる (マイクロ SD イメージファイルの作成方法は付録 A 参考).

「Open a microSD imagefile」ボタンをクリックすると, マイクロ SD イメージファイルを選択するウィンドウが開く. 初期状態では拡張子が「.dmg」のファイルを選択するようになっているが, FAT16 でフォーマットされたイメージファイルであれば, 拡張子の種類は問わない.

ファイルを選択した後, 図 3.2 のようにボタンの下の「Filename」に選択したファイルのパスが表示されたら, ファイルのオープンに成功している. この状態で再度コンソールの RUN ボタンをクリックすると, IPL がマイクロ SD イメージファイルを読み込み,「kernel.bin」という名前で保存されているファイルをカーネルファイルとして実行する.

図 3.2　ファイルがオープンされている状態

3.3　デバッグ機能

3.3.1　ステップ実行

実機と同じようにステップ実行を行うことができる. コンソールの STEP スイッチを上に上げた状態で RUN ボタンをクリックすると 1 命令ずつ実行される.

3.3.2　ブレークポイントの設定

TaC シミュレータでは, レジスタ表示の下にある入力ボックスにブレークしたい命令のアドレスを 16 進数で入力することで設定することで, ブレークポイントの設定ができる. ブレークポイントを設定した後, コンソールの BREAK スイッチを上に上げた状態で RUN ボタンをクリックすると指定した番地の命令を実行した後に停止する. 実機では指定した番地へのメモリアクセスの際に停止するが, TaC シミュレータでは仕様が異なる.

3.3.3 レジスタの内容表示

TaC シミュレータではレジスタの内容を 16 進数で表示している．実機と同じようにロータリースイッチでレジスタを選択し, LED ランプで値を表示することも可能である．表示しているレジスタの情報は, TaC シミュレータが STOP 状態になったときに更新される．

3.3.4 開発者ツールのコンソールへの出力

TaC 実機は USB シリアル通信モジュールと Bluetooth モジュールが接続されており, 接続相手の PC の都合に合わせて, どちらかを選択してターミナルとシリアル通信を行うことができる．TaC シミュレータでは USB シリアルと接続されたターミナルが表示されているので, Bluetooth シリアルはブラウザの開発者ツールのコンソールに接続する．TacOS は, デバッグモードのとき通常の出力を USB シリアルに, デバッグ用メッセージを Bluetooth シリアルに出力するので, TaC シミュレータでデバッグを行う際にはこの機能を使用できる．

この機能を有効にするには,「Enable logger」の横のチェックボックスをクリックする．画面上部のメニューバーから「View」→「Toggle Developer Tools」を選択することで, 開発者ツールが開く．「Console」に移動してから RUN ボタンをクリックすると, 出力が開発者ツールのコンソールにも出力されていることが確認できる．

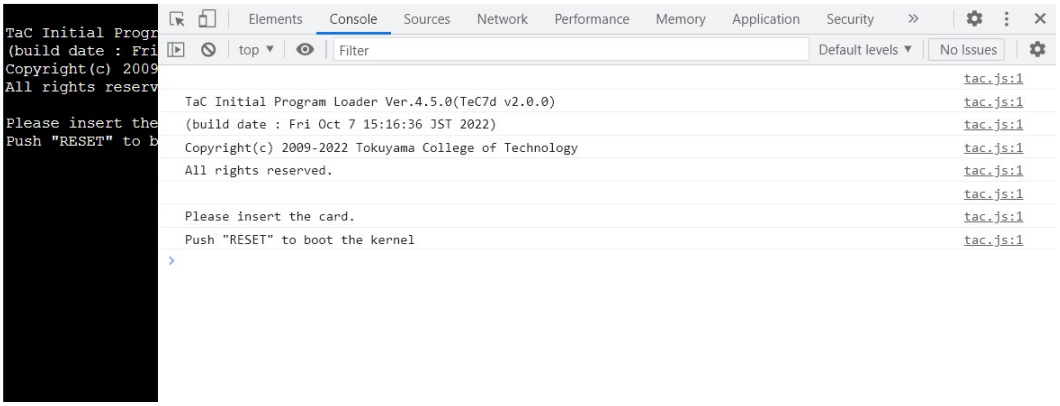

図 3.3 開発者ツールのコンソールへの出力

3.4 試験的機能

ここで紹介する機能は研究のデータ収集用に簡易的に作られたものである．

3.4.1 TLB エントリ拡張

TLB の数とメモリアクセスの速度の関係を調べるために, IO マップの 0x60 から 0x7e に TLB エントリを 8 個新設している．TacOS 側で使用するエントリ数を変更すれば全てのエントリを使用するようになる．

3.4.2 命令の実行回数・TLBMiss の発生回数の計測

この機能は debugmode ブランチで実装している．

指定したプロセス番号で命令が何回実行されているか, あるいは TLBMiss は何回発生しているかを計測することができる．**この機能を使用するには**, TacOS **がプロセスをディスパッチする際にプロセス**

番号を I/O の 0x38 番地に書き込むように修正する必要がある.

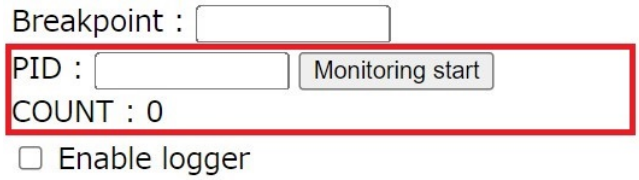

図 3.4　開発者ツールのコンソールへの出力

PID に計測したいプロセス番号を入力し,「Monitoring start」ボタンをクリックすると計測が開始される. もう一度ボタンを押すと, それまでの命令の実行回数と TLBMiss の発生回数が表示される. 図 3.5 では, TacOS の ls コマンドの命令の実行回数と TLBMiss の発生回数を計測している様子である.

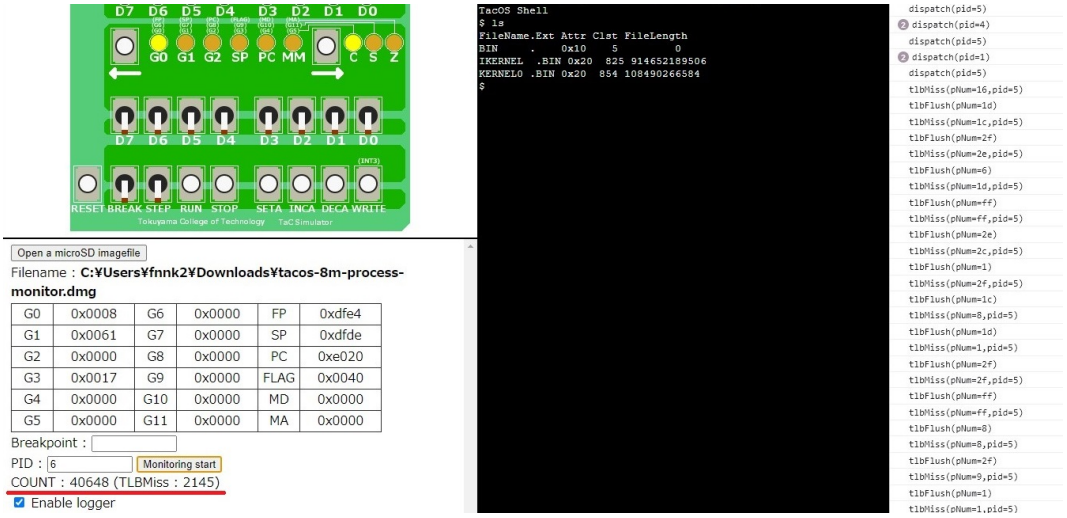

図 3.5　実行回数計測の様子

付録 A

マイクロ SD イメージファイル

A.1 サンプル解説

TaC シミュレータのリポジトリの sample ディレクトリには, サンプルとして 4 つのマイクロ SD イメージファイルが同梱されている.

TacOS-vm-8m.dmg　仮想記憶に対応した TacOS が書き込まれた 8MB のイメージファイル. カーネルと基本的なコマンドの他に sample.cmm という C--言語で書かれたプログラムが入っており, c--コマンドを実行することでコンパイルすることができる.

TacOS-vm-8m-debug.dmg　TacOS-vm-8m.dmg と同じファイルが入っているが, デバッグ用の表示を Bluetooth シリアル経由で出力するようになっている.

Test-8queen-8m.dmg　8 王妃問題を解くプログラムが書き込まれている 8MB のイメージファイル.

Test-SioEcho-8m.dmg　シリアル通信のテストプログラムが書き込まれている 8MB のイメージファイル.

A.2 作成方法

ここでは, macOS 上でマイクロ SD イメージファイルを作成する方法を紹介する.

A.2.1 dmg ファイルの作成

macOS の Launchpad から「ディスクユーティリティ」を実行する. メニューバーから「ファイル」→「空のイメージを作成」を選択すると, 図 A.1 のように, 作成するイメージファイルの設定ウィンドウが表示される. サイズを 8MB 以上 512MB 以下, フォーマットを「MS-DOS(FAT)」, 暗号化を「なし」, パーティションを「単一パーティション - マスターブートレコード・パーティションマップ」にしてから保存. すると, dmg ファイルが作成される.

図 A.1　作成するイメージファイルの設定

A.2.2　パーティションテーブル修正

サイズを 8MB 以上 512MB 以下にしている場合, dmg ファイルのプロパティを見るとファイルシステムが FAT16 であると表示されるが, パーティションテーブルの Type(ファイルシステムの種類) が FAT32 を表す 0x0b になっているため, バイナリエディタを用いて FAT16 を表す 0x06 に修正する必要がある.

マニュアル通りに進めてきた場合, 450 バイト目が Type を表すバイトになっているため, そこを修正すればよい.

```
280  00000000 00000000 00000000 00000000 00000000
336  00000000 00000000 00000000 00000000 00000000
392  00000000 00000000 00000000 00000000 00000000
448  FFFF0BFE FFFF0100 00002FFB 02000000 00000000
504  00000000 000055AA EB3C9042 53442020 342E3400
560  4D452020 20204641 54313620 2020FA31 C08ED0BC
616  6F6E2D73 79737465 6D206469 736B0D0A 50726573
672  00000000 00000000 00000000 00000000 00000000
```

図 A.2　Type の場所

A.2.3 カーネルファイルの格納

TaC シミュレータで使用可能なマイクロ SD イメージファイルを作成できたため, 実行したいプログラムをコンパイルし, kernel.bin という名前で先程の dmg ファイルの中にコピーする (コンパイルの方法は https://github.com/tctsigemura/C--/blob/master/doc/cmm.pdf を参考のこと).

付録 B

TaC シミュレータに関する資料

B.1　I/O マップ

TaC シミュレータの I/O マップを図 B.1 に示す．TaC 実機との違いは赤字で表している．

	+0番地	+1番地	
00h~06h	実機と同じ		タイマー
08h~0Ah	実機と同じ		FT232RL
0Ch~0Eh	空き		TeC
10h~16h	実機と同じ		uSD
18h~1Ch	空き		入出力ポート
1Eh	00H	01H(IN:常にTaCモード)	
20h~26h	空き		
28h	00H	RN4020-Data	RN4020
2Ah	00H	RN4020-Stat/Ctrl	
2Ch	00H	00H	
2Eh	00H	01H(IN:常に接続状態)	
30h~36h	空き		TeCコンソール
38h	00H	参照用PID(OUT)	
…	空き		
60h	TLB[0]上位8bit		MMU
62h	TLB[0]下位16bit		
64h	TLB[1]上位8bit		
66h	TLB[1]下位16bit		
…	…		
9Ch	TLB[15]上位8bit		
9Eh	TLB[15]下位16bit		
A0h	b0=IPL切離し(OUT)		
A2h	b0=MMU有効(OUT) / 違反アドレス(IN)		
A4h	b1=badAddr, b0=memVio(IN)		
A6h	ページ番号(IN)		
…	空き		
F8h~FEh	実機と同じ		コンソール

図 B.1　I/O マップ