Vol.2023-ARC-252 No.24
Vol.2023-SLDM-202 No.24
Vol.2023-EMB-62 No.24
2023/3/24

Jupyter Notebookを介したRISC-V SoC向け実機テスト環境の構築

小島　拓也[1,2,a)]　亀井 愛佳[3]　矢内 洋祐[3]　天野 英晴[3]　久我 守弘[4]　飯田 全広[4]

概要：Society5.0 時代で重要視される MEC(Multi-Access Edge Computing) では, エッジデバイスとクラウドサーバの中間にエッジサーバを配置し, データの機密性向上, 計算処理や通信の低遅延化が期待されている. こうしたエッジサーバでは, 高スループット性, 処理時間の予測可能性および低消費電力性が求められており, 従来の汎用プロセッサを中心とする汎用サーバに変わり, FPGA などの再構成可能ハードウェアを用いたヘテロジニアスな計算機システムの利用が必要である. SLMLET は RISC-V コアと SLM 再構成ロジックで構成される SoC (Syctem-on-a-Chip) で, こうした用途を想定し開発が進められている. 本研究では, 昨年テープアウトされた SLMLET を効率的にテストおよび評価するための環境構築を行い, チップの機能検証と動作条件の測定を自動化した. 電源電圧や動作周波数, テストベクタなど合計 10000 を超える測定条件に関して, 手動の操作を挟むことなく測定することに成功した. 評価結果によれば, テープアウトされた SLMLET チップの RISC-V コア部は 120MHz 動作時に 0.56V で動作可能であることがわかった.

A Chip Testing Methodology for RISC-V SoC Using Jupyter Notebook

1. はじめに

超スマート社会 (Society5.0) の実現に向けて, IoT(Internet of Things) 端末は日々増加しており, その数は 2020 年時点で全世界で 250 億台を超えている [1]. IoT 端末が収集したデータはクラウド・データセンタへ集約され, より良いサービスの提供に向けて利活用される. しかしながら, 従来のクラウド集中型の計算基盤ではスケーラビリティに乏しく, 今後も増加し続けると予想される IoT 端末からの処理要求に対して低遅延で応答するのが困難である.

そこで, 近年は MEC(Multi-Access Edge Computing) と呼ばれる, エッジ側とクラウド側の中間にエッジサーバを配置した計算プラットフォームの採用が進んでいる. エッジサーバは社内ネットワークやローカル 5G ネットワーク上に設置され, 配下のネットワークに接続されたエッジデバイスからの処理を低遅延で処理できることに加え, 閉じたネットワーク内で処理を完結させることができるため, 秘匿性の高いデータを取り扱うサービスにおいてその機密性を向上させることが期待される.

デバイスの多数同時接続を実現しつつ, スマートファクトリーにおける製造装置の自動制御など, 準リアルタイムな処理を行うためにエッジサーバでは高スループット, 低遅延性, さらには処理時間の予測可能性が求められる. 従来の汎用プロセッサで構成される計算機ではこうした要求を満たすことができない. 一方で, 多様化するサービスに追従するには専用ハードウェアを用いるのは設計コストの観点で現実的ではない. そこで, 注目されているのが FPGA(Field Programmable Gate Array) などの再構成可能ハードウェアを用いた計算機システムである. 特に, NIC(Network Interface Card) に FPGA を搭載したいわゆる SmartNIC は, 複雑化するネットワーク通信の暗号化やカプセル化などの処理を FPGA 部に構成したハードウェアで高速に処理を行う. 例えば, FPGA の 2 大ベンダである AMD 社と Intel 社は各社 400G イーサネットポートを備える FPGA を市場に投入している [2], [3].

ところが, これらの多くは PCI-Express 接続のアクセラ

[1] 東京大学大学院情報理工学系研究科
The University of Tokyo
[2] JST さきがけ
JST PRESTO
[3] 慶應義塾大学理工学部
Keio Uniersity
[4] 熊本大学大学院先端科学研究部
Kumamoto University
[a)] tkojima@hal.ipc.i.u-tokyo.ac.jp

Vol.2023-ARC-252 No.24
Vol.2023-SLDM-202 No.24
Vol.2023-EMB-62 No.24
2023/3/24

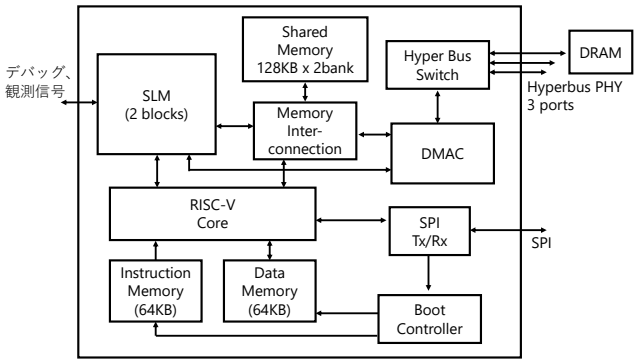


図 1: SLMLET のシステム構成

レータカードであり, サーバーやワークステーションに組み込んで使用されるため, システム全体の消費電力は大きい. また, AMD 社の Zynq シリーズなどの FPGA と CPU を混載した SoC(System on a chip) 型のものもあるが, Linux を稼働できる ARM プロセッサが採用されており用途によっては依然として規模が大きい. そこで, 我々は SLMLET と呼ぶ低消費電力性と低コストに焦点を当てた SoC 型の再構成可能システムの開発を行なっている [4]. SLMLET は再構成ロジックとして SLM (Scalable Logic Module) [5], コントローラとして小規模な RISC-V コア, そして, 相互接続のための Hyperbus インターフェースを持つ. 複数の SLMLET チップを Hyperbus で接続することで, スケーラブルで低消費電力なシステムの実現が可能である.

本稿では, 2021 年にテープアウトされた試作チップのテストおよび評価環境を整備し, RISC-V コア部に焦点を当てた評価結果を報告する. 本環境は, AMD 社の Zynq SoC を搭載した TUL 社の開発ボード PYNQ-Z2 を SLMLET チップのホストとして用いる. PYNQ-Z2 では PYNQ と呼ばれるオープンソースのプラットフォームを利用することで, Python で記述したソフトウェアから FPGA 上のハードウェアを制御することができる. さらに, 配布されている公式のイメージには Jupyter Notebook が含まれており, Web ブラウザ上で Python のコードを実行し, グラフィカルな表示が利用可能である. 本環境を構築するに際して, 以下の実装を行なった:

- SLMLET 制御用 FPGA 設計
- 各種 Python ライブラリ PySLMLET
- コンパイラ, C ライブラリを含む SLMLET SDK

以降の構成は, 2 節にて本研究の評価対象である SLMLET の概要を述べる. 3 節および 4 節ではそれぞれ SLMLET 制御のための FPGA 設計と PySLMLET を用いた実機テストおよび評価環境について説明する. 本環境を用いて得られた評価結果を 5 節で示し, 6 節で本稿の結論を述べる.

2. SLMLET の概要

2.1 システム構成

図 1 に SLMLET のシステム構成を示す. 主な構成モジュールは 1) RISC-V コア, 2) SLM ブロック, 3) Hyper Bus インターフェースである. それぞれを順に説明する.

2.1.1 RISC-V コア

現在の設計では, カリフォルニア大学バークレー校が開発し公開している riscv-mini[6] をベースとした設計を用いている. riscv-mini は Chisel 言語 [7] と呼ばれるハードウェア記述言語で記述されており, 対応する Verilog コードへ変換できる. 基本命令セットである RV32I のみの実装であり, Fetch,Exceute,Write Back の 3 段のパイプライン構成である. ただし, fench 命令はデコードされるものの, 実装はされていない.

SLMLET ではキャッシュは持たず, コア専用のメモリとして各々 64KB の命令用およびデータ用 SRAM が直結されている. プログラムのブートには SPI(Serial Peripheral Interface) を利用し, 命令メモリとデータメモリが外部から送られたプログラムバイナリで初期化される.

2.1.2 SLM ブロック

SLM は論理関数のシャノン展開によって得られる部分関数の特徴を用いることで, 従来の LUT(Look-Up-Table) と比較して省構成メモリ化した FPGA である. SLM 自体の詳細に関しては文献 [5] を参照されたい.

SLM の再構成は共有データメモリに配置されたハードウェア構成情報を RISC-V コアの指示に基づいて SLM ブロックに転送することで行われる. さらに, 圧縮された構成情報を利用することができ, 共有メモリから SLM ブロックに書き込まれる際に伸長される [8]. これによって, 限られた共有メモリの中に多くの構成情報を保持することができる. SLM は 2 つのブロックに分かれており, これに対応する形で共有メモリも 2 つのバンクに分割されている.

SLM のデータ用インターフェースは RISC-V コアのアドレス空間にマップされており, RISC-V コアからメモリのロードストアによってデータの受け渡しが可能である. 加えて, 後述する Hyperbus の DMA コントローラからのストリームデータや SLM 上に実装した回路が共有メモリへ読み書きによって処理データの入出力が可能である.

2.1.3 Hyperbus インターフェース

Hyperbus は JEDEC xSPI に準拠し, Cypress 社が採用してきた 8 ビット幅のシリアルインターフェースである. DRAM として同社の HyperRAM やフラッシュメモリとして同社の HyperFlash などが利用できる. 必要なピン数が少ないため, パッケージの小型化などの利点があり, また汎用 PC で広く利用される DDR メモリと比べると HyperRAM は省電力であることが知られている. SLMLET チップに

Vol.2023-ARC-252 No.24
Vol.2023-SLDM-202 No.24
Vol.2023-EMB-62 No.24
2023/3/24

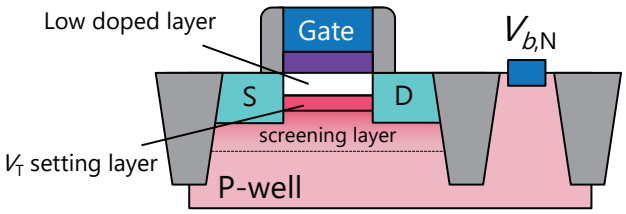


図 2: DDC トランジスタの構造とウェルコンタクト

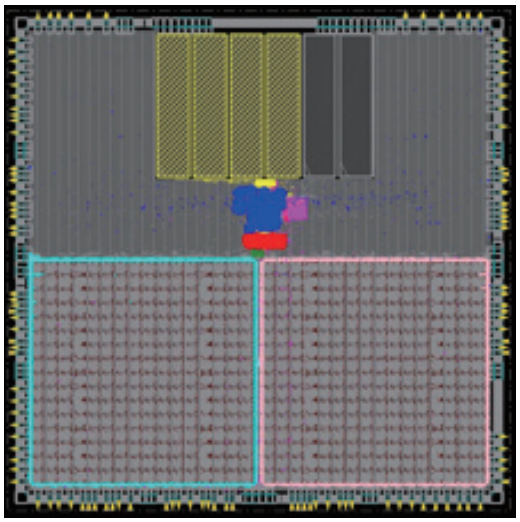


図 3: SLMLET のレイアウト図

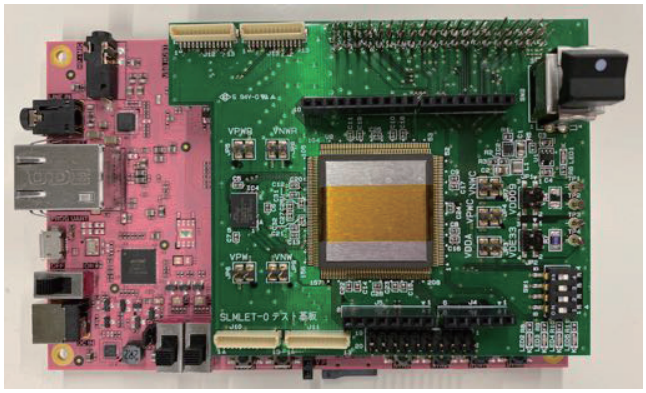


図 4: SLMLET ボードと PYNQ-Z2

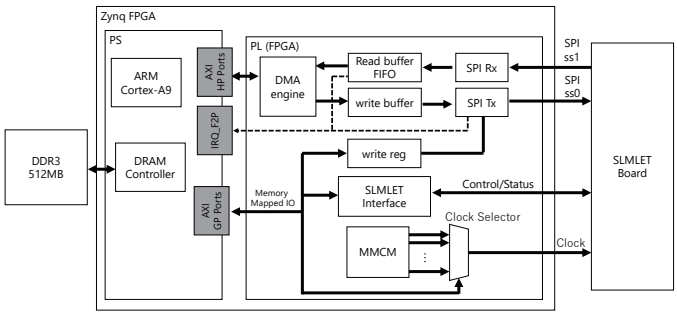


図 5: FPGA 部に設計したハードウェア

は Hyperbus ポートを 3 つ備えており, 互いのチップを相互に接続することや, 前述のメモリモジュールを接続することでシステムの拡張が可能である.

2.2 試作チップの実装

SLMLET の設計は SLM 部を含むそのほぼ全てが Verilog HDL で記述されており, 特殊なハード IP を必要としない. 最初のチップ実装は USCJ DDC 55nm プロセスを採用し, 2021 年にテープアウトが行われた.

DDC(Deeply Depleted Channel)[9] の nMOS トランジスタは図 2 に示す構造をしている. P ウェル上には主に 3 つの異なる層が形成される. ゲート酸化膜直下の low doped layer は不純物濃度を非常し, RDF(Random Dopant Fluctuation) を削減することで閾値電圧のばらつきを抑える. Vt setting layer は複数の閾値電圧を可能にする. こうした特性は, 動作要求の異なるさまざまなブロックで構成される SoC 設計において重要である. Screening layer は高濃度にドーピングされた層で, 空乏層の深さを決定する. さらに, これらの構造は 240mV/V の高い基板効果係数をもたらし, 図中のウェルコンタクトに印加するバイアス電圧を変化させることで閾値電圧を調整し, リーク電流の削減などが可能である.

SLMLET のロジック部は C55DDCT07L60LVT(セル高: 7 トラック, ゲート長 60nm, Low voltage threshold) のスタンダードセルライブラリを使用して, Synopsys 社 Design Compiler で論理合成を行なった. レイアウトは Candence 社 Innovus を使用した. テープアウトされたレイアウトを図 3 に示す. チップサイズは 4.2mm 角である.

2.3 評価ボードの開発

パッケージされた SLMLET チップと制御および観測用 FPGA を接続するための評価ボードを開発した. FPGA には前述の PYNQ-Z2 ボードを採用している. PYNQ-Z2 には GPIO として Raspberry Pi 40 ピンヘッダと Arduino ヘッダがあり, ここに刺さるように設計されている. 図 4 に PYNQ-Z2 へ SLMLET ボードを装着した様子を示す.

Hyperbus インターフェースの 1 ポートには表面実装された Infineon 社の 64Mbit HyperRAM を接続し, 残りの 2 ポートはボード上のコネクタを通して他の SLMLET ボードと接続される. その他, メモリマップトな LED および DIP スイッチが設けられている.

3. FPGA 部の設計

本節では, PYNQ-Z2 の FPGA 部を用いて SLMLET を制御および観測するためハードウェア設計を説明する. PYNQ-Z2 を含む, Xilinx 社の Zynq シリーズには PS(Processing System) と呼ぶ CPU 部と, PL(Programmable Logic) と呼ぶ FPGA が存在し, 互い

が直接接続されている．PS 部では Linux OS が稼働し, PYNQ と呼ばれるソフトウェアスタックを用いることで, PL 部に設計したハードウェアはラップされた各種 Python モジュールで容易に制御可能である．

PL 部に設計されるハードウェアと PS 部がコミニュケーションする場合は, AXI ポートで接続する．本研究で設計したハードウェアでは主に SLMLET を制御するためのモジュールとデータ転送のためのモジュールに分けられ, それらは AXI Interconnect を介して PS 部と接続されている．図 5 に設計したハードウェアの構成を示す．

3.1 SLMLET の制御

SLMLET の制御は, リセット, ブートの開始などを含む．これを PS 側から制御するために, これらの制御信号に対応するメモリマップトレジスタを用意している．また, SLMLET を駆動するためのクロック信号も FPGA 内部の MMCM(Mixed-Mode Clock Manager) と呼ぶクロック生成器から送る．ただし, SLMLET をテストするクロック周波数を変更するたびに FPGA を再構成するのを避けるために, 複数の異なるクロックを生成し, それらを PS からの選択信号で切り替えられるクロック選択回路を設計している．選択回路は FPGA の DRC エラーを避けるために, FPGA 上の BUGCTRL というクロック専用の資源をカスケード接続して実現している．今回採用している PYNQ-Z2 の FPGA では利用可能な BUFGCTRL の数に限りがあるため, 一つの設計で利用可能なクロック数は 12 種までである．

3.2 SLMLET とのデータ転送

データ転送には, RISC-V コア動作開始前のブートロードおよび, コア動作中のデータ送受信に分けられる．ブート用のプログラムバイナリは, まず PS に接続された DRAM 上にロードされ, DMA 転送によって FPGA 上の write buffer に転送される．SLMLET のコア専用メモリは命令用とデータ用合わせて 128KB であり, write buffer もこのサイズと同じになっている．write buffer に転送が完了すると, ブート開始に対応する制御レジスタに 1 を書き込むことで SPI を介してデータが SLMLET へ送られる．すべてのデータが無事に転送し終わると, PS 側へ割り込み信号が通知される．同様に, コア動作中に FPGA からデータを送る場合は設定レジスタ write reg にデータを書き込むことで行われる．ただし, SPI は SLMLET 側がマスターであるため, 送信許可が出ない限りこれらの転送は行われない．

SLMLET から SPI を介して送信されたデータは read buffer に格納される．これはキュー (FIFO) になっており, データ数が事前に設定した値以上になると割り込み信号が PS 側へ通知される．キューのサイズはパラメータライズされているが, デフォルトでは 4KB になっている．SLMLET は最大で 4 つの SPI スレーブを接続でき, 本実装では送信の方向を区別するために, FPGA から SLMLET のデータ転送にチップセレクト信号 0, その反対をチップセレクト信号 1 を用いて行う．

表 1: FPGA のリソース消費

資源	使用量	利用可能量	使用率 (%)
LUT	6128	53200	11.5
LUTRAM	1227	17400	7.05
FF	7925	106400	7.45
BRAM	68	140	48.6
IO	25	125	20.0
BUFG	27	32	84.4
MMCM	2	4	50.0

3.3 動作速度とリソース消費

本設計は最大で 130MHz 程度までタイミング違反なしで実装が可能であったが, 安定的に SLMLET とのデータ転送を行うために, 20MHz の動作クロックで実装している．この時のリソース消費を表 1 に示す．前述の通り, BFUG を多く使用しているが, LUT などその他のロジック部には十分が余裕がある．また, BRAM に関しても 50%程度の消費であるため, read buffer のサイズをデフォルトの 4KB からより大きいサイズに変更する余地が残っている．

4. PySLMLET を用いたテスト環境

3 節で述べた FPGA 上のハードウェアを制御し, SLMLET チップのテストおよび評価を行うための Python ライブラリ “PySLMLET” について説明する．本ライブラリは以下の要素に分けられる:

- FPGA 上ハードウェアのドライバ
- プログラムバイナリ用ユーティリティ
- VISA/SCIP 対応直流電源制御ライブラリ
- 自動テストフロー
- 操作用 GUI

これらのライブラリは PYNQ-Z2 ボード上にインストールされ, Jupyter Notebook またはターミナルなどから利用できる．

4.1 ハードウェアドライバ

PYNQ の DefaultIP クラスを継承した Python クラスを定義し, FPGA 上の各種自作モジュールを制御するためのドライバを実装した．例えば, SLMLET Interface を扱うための SLMLETInterfaceDriver クラスのインスタンスには reset() メソッドがあり, これを呼ぶだけで SLMLET のシステムリセットが完了する．このように各種操作がラップされている．

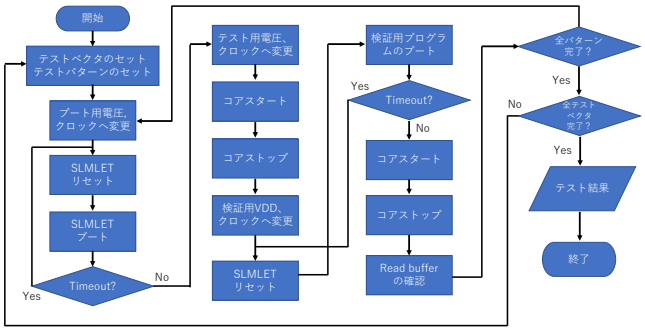


図 6: テストのフロー

4.2 プログラムバイナリ用ユーティリティ

SLMLET の RISC-V コアで実行されるプログラムを用意する方法として 1) C ソースコード, 2) クロスコンパイル済み ELF ファイル, 3) ブート用メモリダンプの 3 通りを用意し, それぞれに対応するクラスを実装した. これらのインスタンスは共通の boot 用のメソッドに引数として渡される. C ソースコードからプログラムバイナリを用意するには RISC-V 用コンパイラが PYNQ-Z2 上にインストールされている必要があるが, 別の PC などでクロスコンパイルを行う必要がなく, PYNQ-Z2 上でアプリケーションの開発が完結する. 一方で, PYNQ-Z2 とは別の環境でコンパイルされた ELF ファイルやそこからブートで送られるデータを抜き出したメモリダンプを直接利用することもできる.

4.3 VISA/SCIP 対応直流電源制御ライブラリ

VISA(Virtual Instrument Software Architecture) とは電源装置やオシロスコープなどの装置を制御するための共通規格化されたライブラリである. これらに対応した装置は Python からは PyVISA などのライブラリを用いて, 制御することができる. 本ライブラリでは, テスト環境においてこれに対応した安定化直流電源と連携できるようライブラリを整備した.

これらの装置を制御するには SCPI(Standard Commands for Programmable Instruments) と呼ばれるコマンドフォーマットを用いることが多く, 本実装もこれを想定している. 現時点で, RIGOL 社の DP932 シリーズや Keysight 社の E3631A との動作を確認している. PyVISA のバックエンドとしては各ベンダーが提供するものではなく Python のみで記述された PyVISA-py を用いている. これは, 一般にベンダーから提供されるものは Windows 向け, かつ x86-64 プロセッサ向けであることが多く, ARM プロセッサで Linux OS の PYNQ-Z2 の環境とは互換性がないためである.

4.4 自動テストフロー

ここまでに説明したライブラリにより, ソフトウェアから SLMLET に入力されるクロック信号の変更や, 印加する電源電圧の変更が可能となった. これを利用して, テストベクタや周波数, 電圧などの条件変更して自動でテストするルーチンを用意した. テストのフローを図 6 に示す構造をしている.

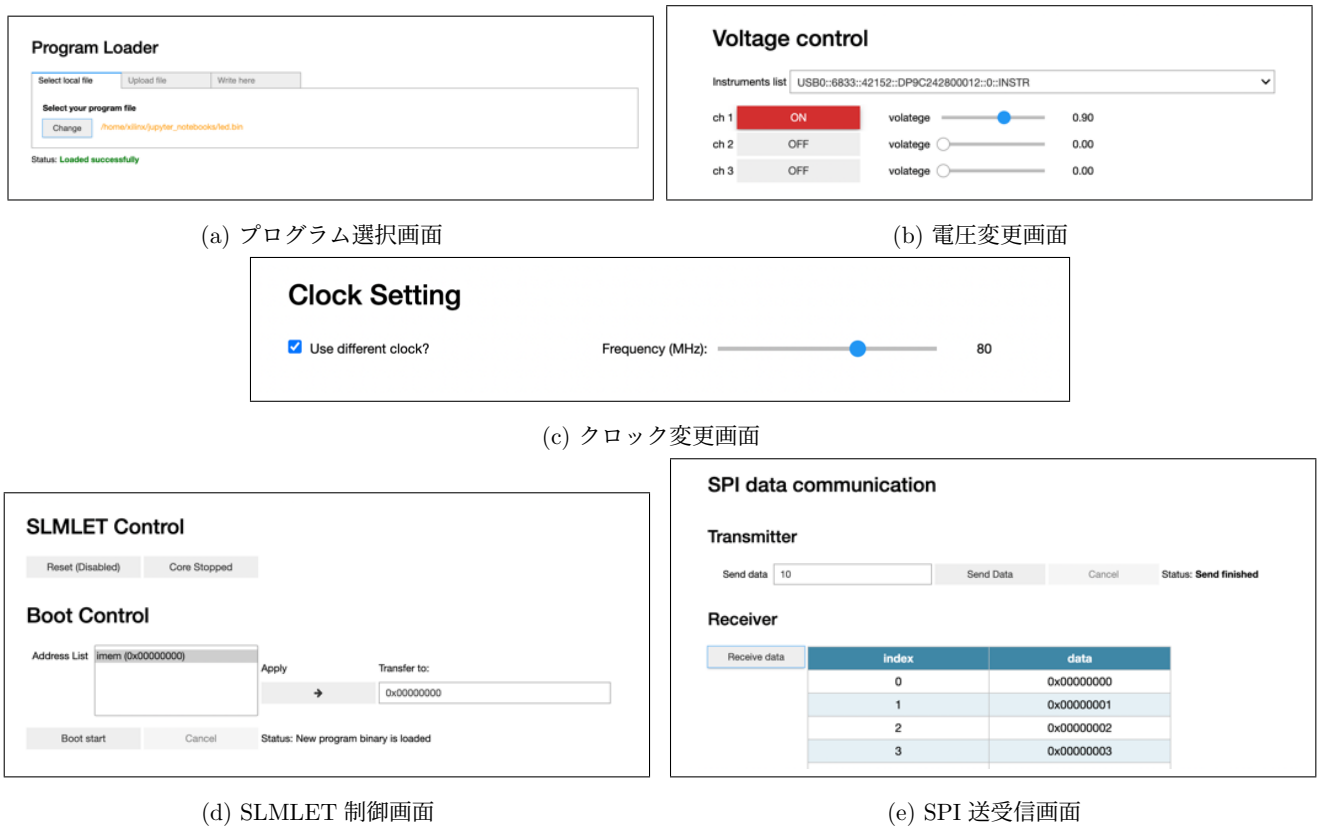
対象のプログラム (テストベクタ) が正しく動作したかどうかはデータメモリに書き込まれた値を確認して判断する. ゆえに, テスト用のプログラム実行とデータメモリの内容を SPI を通してダンプするだけのプログラム (検証用プログラム) の二つを連続して実行することになる. このとき, ブートを含む SPI を介したデータ転送が正しく行われていることが必須である. そこで, これらの操作には十分にテストが行われた電圧と周波数が利用される. ただし, もしブート時に SLMLET が正しく応答せず SPI の送信許可が出なかった場合は, 転送が完了せず, 終了を通知する割り込みが発生しない. そのため, 一定の待ち時間内にブートが完了できなかった場合は失敗とみなし, リセットから再開する. 図に示す通り一つのプログラムをテストおよび検証するのに電圧やクロックの変更が 3 回行われる. これらの一連の処理を全ての動作条件とテストベクタについて検証が完了すると, 最終的にテスト結果を保存する.

4.5 操作用 GUI

PYNQ では Jupyter Notebook が利用できる利点を活かし, これまでに説明した機能を GUI で利用できるよう整備した. 図 7 に各操作に対応する GUI を示す. 図 7(a) は SLMLET にロードするプログラムを選択する画面である. PYNQ-Z2 の SD カード内に保存された RISC-V バイナリや C ソースコードを選択できるのに加え, Jupyter Notebook を開いている PC からネットワーク経由でアップロード, さらには, 直接テキストエリアに C コードを記述してその場でコンパイルすることもできる.

図 7(b) は PYNQ-Z2 に直接接続された VISA 対応の直流電源を操作する画面である. 複数のチャネルを持つ電源装置であれば各チャネルごとに電圧を設定するスライドバーが現れる. また, チップ測定に用いる電圧はたかだか 1V 強であるが, これらの装置は 30V ほどの電圧まで出力できる場合が多く, 操作ミスによるチップの破壊などの恐れがある. この GUI では設定可能な電圧範囲に制限を設けており, こうした操作ミスをある程度抑制できると期待される.

図 7(c) は SLMLET に入力するクロック信号を選択する画面である. 図 7(d) は SLMLET のリセット, コアの動作開始および停止, ブートの設定と開始のための画面である. ブートの設定では, 読み込んだプログラムデータをどの番地に書き込むかを指定する. ELF ファイルまたは C ソースコードからプログラムデータを用意した場合はシンボルテーブルが利用できるため, それらの情報がアドレスリス



(a) プログラム選択画面　(b) 電圧変更画面

(c) クロック変更画面

(d) SLMLET 制御画面　(e) SPI 送受信画面

図 7: Jupyter Notebook 上の各 GUI

トに表示される.

図 7(e) はコア動作中に SPI を介したデータの送受信を行う画面である. 上側は SPI の送信を行う部分で, Send Data ボタンを押すとテキストボックスに入力されたデータが SLMLET に送信される. ただし, SLMLET が読み出し状態に移行し, SPI の送信許可が出ない限り送信が完了しないため, 途中で送信を取りやめるためのキャンセルボタンが備わっている. 下側は Read buffer に入っているデータを読み出すための画面で, ボタンを押すとその時点で入っている全てのデータを読み出しテーブルで表示する. もし, Read buffer が空の場合は何も表示されない.

また, これらと同様のインターフェースを用いて自動テストフローの条件を設定する GUI や結果の可視化機能も併せて実装している.

5. テストと評価の結果

本節では, 3 節の FPGA 設計ならびに 4 節のライブラリを用いて, SLMLET の RISC-V コア部の機能検証と動作条件を測定する. 機能検証には RISC-V ISA をテストするために公開されている riscv-test[10] を用いる. riscv-test には RV32I 命令セットの各命令をテストする合計 39 種のテストベクタが含まれる. ただし, 今回 SLMLET をテープアウトするのに用いた RTL には利用した Chiesl バージョン 3.4.4 のバグにより lh 命令および lhu 命令が正しく動作しない Verilog コードが使用されていることが判明した. そのため, 39 種のうち, これらの 2 命令が利用される *rv32i-p-lh,rv32i-p-lhu,rv32i-p-sh,rv32i-p-sb* および未実装の fence 命令に関する *rv32i-p-fence_i* の 5 つを除く 34 種をテストに用いる. riscv-test の元々の実装では, テストの結果を所定のメモリ番地に書き込むようになっている. 本テストでは, この書き込み先番地を SLMLET のコア専用データメモリになるよう変更を加え, ブートロード時にその領域はゼロで初期化されるようする. 検証用プログラムでは, この番地から値を読み出し, SPI で FPGA 側へメモリ中のデータをダンプする.

SPI の制御用のプログラムなどはライブラリとして整備し, コンパイル時にリンクされる. また, SLMLET のメモリマップに併せてリンカスクリプトを用意し, 正しいプログラムバイナリが生成できるようにした. これらをまとめて SLMLET SDK としてまとめ, PYNQ-Z2 上で利用できるようにした.

5.1 機能検証

動作周波数を 20MHz, 電源電圧を標準の 0.90V に設定し前述の 34 個のテストを実行した. その結果, 全てのテストに合格し, RISC-V コアのレイアウト上の致命的なバグはないことが確認できた.

Vol.2023-ARC-252 No.24
Vol.2023-SLDM-202 No.24
Vol.2023-EMB-62 No.24
2023/3/24

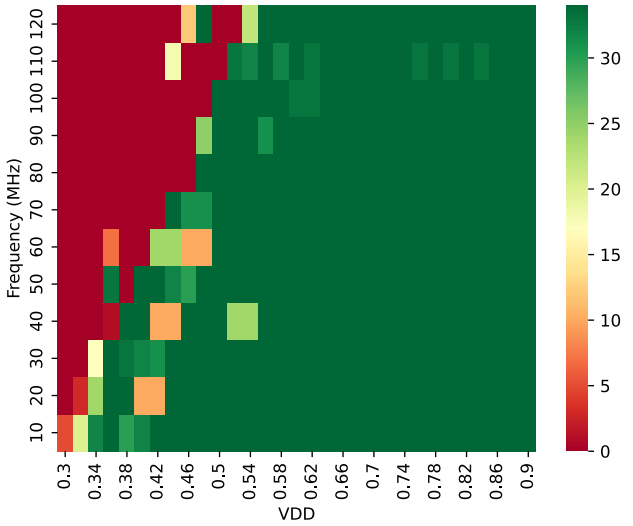


図 8: 動作範囲の測定結果

5.2 動作範囲

次に, バイアス電圧を標準状態にし, 電源電圧 VDD を 0.30V から 0.90V まで 0.02V 刻みで, 周波数を 10MHz から 120MHz まで 10MHz 刻みで変化させ, すべてのテストを実行した. 検証パターンは 12648 ケースにのぼるが, 本環境で実装した自動フローによって一度も手動の操作を挟むことなくテストを実行することができた.

図 8 に各条件でパスしたテストの数をヒートマップ化したものを示す. 概ね, 電源電圧に比例して動作可能な最高周波数が上がっていることが確認できる. 10MHz では 0.36V, 120MHz でも 0.56V ですべての命令が動作することが確認できた. 一部直流電源または測定フローの問題のためか, 動作を確認した電圧よりも高い電圧で失敗するケースが見受けられた. この原因に関しては今後明らかにしていく予定である.

6. おわりに

本稿では PYNQ-Z2 ボード上で動作する SLMLET のためのテストおよび測定環境を構築した. これには PYNQ-Z2 上の FPGA 設計, 各種制御用 Python ライブラリ, コンパイル環境を含む SLMLET 向けソフトウェア開発環境が含まれる. また, 条件を変更し自動でテストを行う機能や, 視覚的な操作が容易にできるような GUI を整備した. 本環境で得られた SLMLET の RISC-V コア動作範囲は概ね予想されていた結果と一致したものの, 一部不可解なテスト結果となる条件が存在した. これらの原因の究明やテストの安定化を図る必要がある. さらに, 本報告ではボディバイアス電圧を標準状態としていたが, これらの変更も条件に加え, ボディバイアス効果の特性を確認し, より電力効率の良い動作条件の探索を行う必要がある.

謝辞

本研究は東京大学大規模集積システム設計教育研究センターを通しシノプシス株式会社, ケイデンス株式会社, ならびにメンターグラフィックス株式会社にの協力で行われたものである. また, 本研究は科学技術振興機構戦略的研究推進事業 (JST) CREST JPMJCR19K1 および JST さきがけ JPMJPR22P5 の支援を受けたものである.

参考文献

[1] 総務省：令和 3 年度版情報通信白書, https://www.soumu.go.jp/johotsusintokei/whitepaper/ja/r03/pdf/index.html (2021).
[2] Ganusov, I. K., Iyer, M. A., Cheng, N. and Meisler, A.: Agilex™ generation of intel fpgas, *2020 IEEE Hot Chips 32 Symposium (HCS)*, IEEE Computer Society, pp. 1–26 (2020).
[3] Dastidar, J., Riddoch, D., Moore, J., Pope, S. and Wesselkamper, J.: AMD 400G Adaptive SmartNIC SoC: Technology preview, *2022 IEEE Hot Chips 34 Symposium (HCS)*, IEEE Computer Society, pp. 1–31 (2022).
[4] 矢内洋祐，小島拓也，　奥原颯，天野英晴，飯田全広ほか：RISC-V MP および SLM 再構成ロジックを混載した「SLMLET」チップの予備評価，研究報告システム・アーキテクチャ (ARC)，Vol. 2022, No. 11, pp. 1–6 (2022).
[5] Amagasaki, M., Araki, R., Iida, M. and Sueyoshi, T.: SLM: A scalable logic module architecture with less configuration memory, *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. 99, No. 12, pp. 2500–2506 (2016).
[6] UC Berkeley Architecture Research: riscv-mini, https://github.com/ucb-bar/riscv-mini.
[7] Bachrach, J., Vo, H., Richards, B., Lee, Y., Waterman, A., Avižienis, R., Wawrzynek, J. and Asanović, K.: Chisel: constructing hardware in a scala embedded language, *DAC Design automation conference 2012*, IEEE, pp. 1212–1221 (2012).
[8] 高木颯平，丹羽直也，四釜快弥，矢内洋祐，天野英晴，中里優弥，尼崎太樹，飯田全広ほか：SLM 細粒度再構成ロジックにおける構成情報の圧縮，研究報告組込みシステム (EMB)，Vol. 2022, No. 11, pp. 1–6 (2022).
[9] Fujita, K., Torii, Y., Hori, M., Oh, J., Shifren, L., Ranade, P., Nakagawa, M., Okabe, K., Miyake, T., Ohkoshi, K. et al.: Advanced channel engineering achieving aggressive reduction of V T variation for ultra-low-power applications, *2011 International Electron Devices Meeting*, IEEE, pp. 32–3 (2011).
[10] RISC-V Software: riscv-mini, https://github.com/riscv-software-src/riscv-tests.