一般社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
IEICE Technical Report
RECONF2023-31(2023-09)

FPGA と RISC-V プロセッサを搭載した SoC 向け HW/SW 設計フローと実機評価

小島　拓也† 矢内　洋祐†† 奥原　颯††† 天野　英晴†† 久我　守弘†††† 飯田　全広††††

† 東京大学 情報理工学系研究科
†† 慶應義塾大学理工学部
†† シンガポール国立大学
†††† 熊本大学大学院先端科学研究部
E-mail: †tkojima@hal.ipc.i.u-tokyo.ac.jp, ††{sodium,hunga}@am.ics.keio.ac.jp, †††hayate01@nus.edu.sg, ††††{kuga,iida}@cs.kumamoto-u.ac.jp

あらまし　SLMLET は RISC-V コアと SLM 再構成ロジックで構成される低消費電力な SoC (Syctem-on-a-Chip) で, エッジコンピューティング向けのデバイスとして期待される. 本研究では, FPGA CAD と連携したアプリケーション開発環境やライブラリを実装し, 実機測定に基づく評価を実施した. 計算負荷の高い処理を SLMLET の FPGA 部に実装した専用回路にオフロードすることで, 計算に必要なサイクル数を削減し, SLMLET の RISC-V コアでのみ計算する場合や, 市販のエッジ端末向けデバイスである ESP32 や Raspberry Pi Pico によるソフトウェア処理と比べレイテンシを 40% 程度まで削減し, 消費エネルギーも最大で 80% ほど削減できることを示した. さらに, プロセッサと FPGA が SoC 上で密結合していることにより, オフロードのオーバーヘッドが削減でき, ディスクリート型の FPGA を使用するの比べ, 1 桁から 2 桁ほど消費エネルギーを削減できる可能性を示した.

キーワード　FPGA, RISC-V

1. はじめに

Society 5.0 の実現に向け, コンピュータには高度な情報処理能力とリアルタイム性が求められる. 加えて, エネルギー効率も重要視される. 例えば, エッジコンピューティング環境においては, エッジデバイスがデータを収集し, それらを分析することで, 何らかの意思決定を行う. このとき, エッジデバイスの電力源としてバッテリーを用いていることも多く, 消費電力の削減が必須である.

こうした要求に対して, Field-Programmable Gate Array(FPGA) は有望な解決策の一つである. FPGA はプログラマブルなハードウェアであり, 目的に特化した計算回路をプログラムすることで, 高効率に処理を行うことが期待される. とりわけ, エッジ用途では組み込み FPGA IP(embedded FPGA: eFPGA) をマイクロプロセッサなどと同じ製造プロセスで一つのシリコンダイに混載する System-on-a-Chip(SoC) が盛んに研究されている [1]. eFPGA の用途としては, 入出力インターフェースとプロセッサとの間のグルーロジックや暗号処理などの高負荷な処理をオフローディングする回路として用いる. 例えば, 面積制約から各種暗号方式の専用回路が搭載できない場合, 必要な暗号方式の回路を FPGA 上で切り替えながら利用することができる. さらに, プロセッサと FPGA がチップ内で密に結合されているため, これらの間でのデータ転送に関する性能面と電力面のオーバーヘッドが小さい.

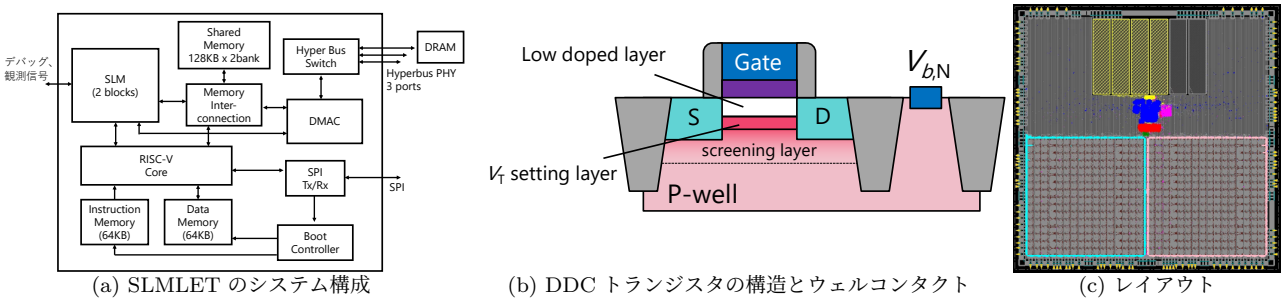
eFPGA においては, 省面積および省電力という特徴が重要である. そこで, 従来の Look-Up-Table(LUT) 方式よりも構成メモリが削減可能な Scalable Logic Module(SLM) を論理セルとして用いた FPGA が提案されている [2]. 本稿では, SLM ベースの FPGA と小規模なマイクロプロセッサとして RISC-V コアを搭載した SoC “SLMLET” を対象とし, ライブラリを含む, FPGA 設計 CAD と連携したアプリケーション開発環境の構築と実機評価の結果を報告する.

2. SLMLET の概要

2.1 システム構成

SLMLET は図 1(a) に示すように, RISC-V コア, SLM ブロック, および各種 SRAM メモリを構成要素として持つ SoC である. RISC-V コアは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. SLMLET ではキャッシュではなく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 を持つ. プログラムのブート時には Serial Peripheral Interface(SPI) を介して送られるプログラムバイナリで命令メモリとデータメモリが初期化される.

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 と比較して構成メモリが小さい. SLMLET は SLM を用いた FPGA を 2 ブロック持ち, それらを協調利用して一つの大きなブロックとして用いることや, それぞれを独立に用いることもできる. 各ブロックはタイルを 2 次元配列状に並べた構造を持ち, タイルは階層化された構造となっている. 各タイルには Logic Block (LB) とタイル間の相互接続網のためのスイッチなどで構成される. LB は Basic



(a) SLMLET のシステム構成　(b) DDC トランジスタの構造とウェルコンタクト　(c) レイアウト

図 1: SLMLET チップの実装

Logic Element (BLE) を複数束ねたクラスタであり, BLE は一つの SLM とフリップフロップ (FF) で構成される. タイル数や LB 内の BLE 数などは IP 生成ツールでパラメータ化されている [2]. SLMLET では LB あたり 4 つの BLE を持ち, BLE には 5 入力 SLM を採用した IP が用いられている.

2.2 試作されたチップ

SLMLET の設計は 2021 年にテープアウトされた. 試作に用いた USCJ DDC (Deeply Depleted Channel) 55nm プロセス [3] は図 1(b) に示す構造 (nMOS トランジスタ) 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 Cadence 社 Innovus を使用した. テープアウトされたレイアウトを図 1(c) に示す. チップサイズは 4.2mm 角である. レイアウト下側の線で囲われた 2 つの矩形領域が SLM ブロックである.

3. HW/SW 設計フローとライブラリの実装

本節では, 本研究で開発した SLMLET 向けのアプリケーション開発フローについて説明する. アプリケーションの実行バイナリが生成されるまでのフローを図 2 に示す. 図に示す通り, FPGA CAD [2] を用いた SLM 部のハードウェア開発設計フローと SW 部のコンパイルフローに分けられる. FPGA CAD では一般的に利用される Yosys(論理合成), ABC(テクノロジマッピング), VPR(クラスタリング, 配置) に加えて, 配線には EasyRouter[4] が利用されている. この FPGA CAD を用いて生成されるビットストリームは, SLM ブロック一つ分のコンフィギュレーションデータであり, 14616 バイトの固定長である. 生成されたビットストリームファイルは後述するバイナリフォーマットへ変換され, 実行ファイルに組み込まれる. 複数種類のビットストリームを利用可能にするために, 各ビットストリームはバンドルされ一つのオブジェクトファイルへ変換される.

RISC-V 部で実行されるソフトウェアは, C 言語でプログラムされる. C 言語のソースファイルは GCC でコンパイルされ, SLMLET 用ライブラリおよびバンドル化されたビットストリームファイルと共にリンクされ実行バイナリが生成される.

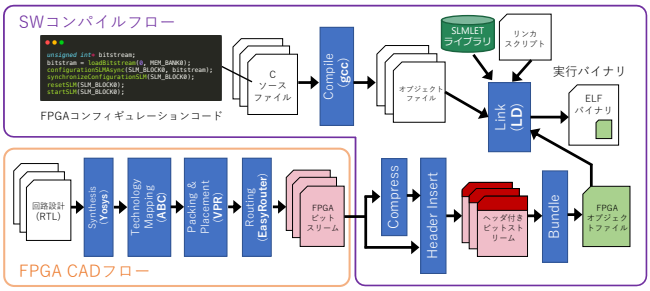


図 2: RISC-V プロセッサ (SW) および SLM 部 (HW) の設計フロー

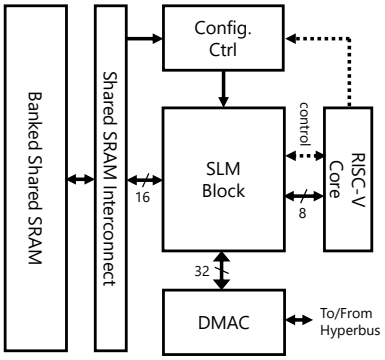


図 3: SLM ブロック IO と周辺モジュール

3.1 SLMLET ライブラリ

SPI の送受信や printf, scanf などの標準入出力, HyperRAM とのデータ転送などは静的ライブラリとして実装されており, プログラムから対応する関数を呼び出すことで容易に利用できる [5]. 本研究では, SLM 部の再構成やデータ転送の機能を追加し, ライブラリの拡張を行なった.

SLMLET における SLM ブロックの IO とその他のモジュールとの接続関係を図 3 に示す. コンフィギュレーションコントローラは RISC-V コアからの指示に応じて SLM ブロックにビットストリームデータを書き込み, 再構成を行う. 再構成にあたり, ビットストリームデータは共有 SRAM 上においておく必要があるが, RISC-V コアとは独立に動作可能であるため, 再構成の最中も RISC-V コアは別の処理を続けることができる. また, SLMLET では Tag-Less Compression(TLC)[6] により事前に圧縮されたビットストリームデータを On-The-Fly で伸長しながら再構成することもできる. 最終的に SLM ブロックに書き込むデータ量は変化しないが, メモリ上に保持しておくべきビットストリームのデータサイズを削減できる. その他, RISC-V はメモリマップトな制御レジスタを介して SLM ブ

ロックに対して，リセット信号などを送る．

SLM ブロックへデータを転送するインターフェースが 3 種類用意されている．1 つ目は RISC-V コアから直接読み書きが可能なインターフェイスであり，RISC-V コアからはメモリマップトな領域として扱われる．8bit のデータ信号に加え，8bit のアドレス信号およびハンドシェイクのための ready, valid 信号などを備える．2 つ目は SLM ブロックが能動的に共有 SRAM への読み書きを行うインターフェイスで，1 度に 16bit 幅のデータアクセスが可能である．3 つ目は DMA コントローラと直結したインターフェイスで，HyperBus から転送されるストリームデータを扱う．

実装したライブラリには RISC-V から SLM ブロックやコンフィギュレーションコントローラの制御，RISC-V コアと直結したインターフェイスとのデータ転送を行う関数が含まれる．代表的な関数は以下の通りである．

（1） `unsigned int getBitstreamCount()`
実行ファイルに組み込まれているビットストリームの数を返す．

（2）
```
void* loadBitstream(unsigned int index,
        t_mem_bank bank)
```
`index` 番目のビットストリームをコア専用データメモリから共有 SRAM の `bank` 番目のバンクにロードする．ロードされたビットストリームの先頭をポインタで返す．

（3）
```
void configurationSLM(t_slm_block block,
    void *bitstream)
```
共有 SRAM 上にロードされたビットストリームで `block` 番目の SLM ブロックを再構成する．

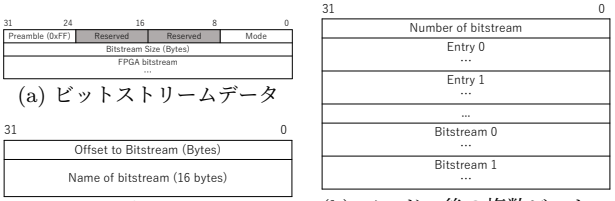
（4） `void resetSLM(t_slm_block block)`
`block` 番目の SLM ブロックにリセット信号を送る．

（5） `void startSLM(t_slm_block block)`
`block` 番目の SLM ブロックをアクティブにする．

（6）
```
void writeSLM(t_slm_block block,
    void *offset, TYPE data)
```
`block` 番目の SLM ブロックのメモリマップト領域にデータを書き込む．

（7）
```
TYPE readSLM(t_slm_block block,
    TYPE *offset)
```
`block` 番目の SLM ブロックのメモリマップト領域からデータを読み出す．

再構成を行う `configurationSLM` は再構成が完了するまでブロッキングする関数であるが，このほかに，ノンブロッキングの `configurationSLMAsync` なども備わっている．RISC-V コアと直結したインターフェイスからデータを転送する場合，データ型に応じて必要な処理が変わる．例えば，32bit の int 型データを転送する場合は，8bit のインターフェイスを介して合計 4 回のデータ転送が必要である．`writeSLM` および `readSLM` 関数は実際のところ Generic を用いたマクロとなっており，渡された変数の型 *TYPE* に応じて適切な回数のデータ転送が行われるようになっている．



(a) ビットストリームデータ

(c) バンドル内のエントリーフォーマット

(b) バンドル後の複数ビットストリーム

図 4: バイナリフォーマット

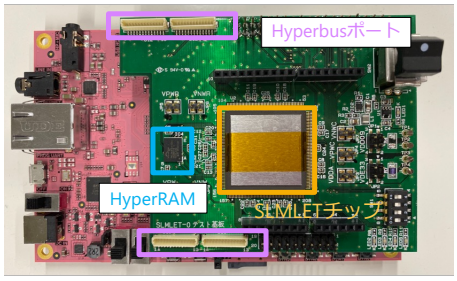


図 5: SLMLET ボードと PYNQ-Z2

3.2 ビットストリームのバイナリフォーマット

前述のライブラリ関数を実装するにあたり，図 4 に示すビットストリームデータのバイナリフォーマットを定義した．まず，FPGA CAD で生成したビットストリームデータは，図 4(a) に示すヘッダが付加される．Preamble は有効なビットストリームデータが開始することを意味し，Mode はビットストリームの圧縮，非圧縮を区別するために使用されるフィールドである．前述の通りビットストリームは通常は固定長であるが，圧縮時には可変長なるため，ビットストリームのバイト数がヘッダに埋め込まれている．

ビットストリームの圧縮は図 2 に示す通り，コンパイルオプションとして行われる．そのため，RISC-V コアのソフトウェアから圧縮，非圧縮を明示的に指示する必要はなく，ライブラリがこのヘッダを見て判断する．

SLMLET では SLM ブロックが 2 つ存在し，また，時分割的に再構成を行い複数の回路を切り替えて利用したい場合など，1 つのアプリケーションは複数のビットストリームを扱える必要がある．そこで，コンパイル時に必要なヘッダが付加されたすべてのビットストリームデータを 1 つのバンドル化されたオブジェクトにまとめられる．この際にオブジェクトにはまた新たなヘッダが 1 つ付加される．そのフォーマットを図 4(b) に示す．ヘッダの先頭は合計でいくつのビットストリームデータが含まれるのかを示すフィールドから始まり，その後，固定長の Entry がビットストリームの数だけ続き，各ビットストリームが連結される．Entry は図 4(c) に示すように，オブジェクトファイルの先頭からビットストリームデータ開始位置である offset とビットストリームの名前 (最大 16bytes) を保持する．ただし，ビットストリームデータの名前は SLM ブロックの再構成に必要なものではなく，あくまでデバッグ用途などで用いられる．

SLMLET 用に用意したリンカスクリプトでは，このオブジェクトファイルをデータメモリの bss セクションの後に配置し，開始位置に所定のシンボルを埋め込む．ライブラリはこのシンボル位置を参照し処理を行う．

4. 評価ボードと測定ソフトウェア

図 5 のようなパッケージされた SLMLET チップと制御および観測用 PYNQ-Z2 ボードを接続するための評価ボードが

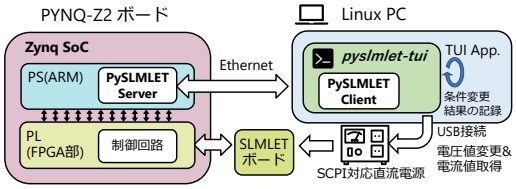


図 6: 自動測定環境の概要

開発されている．PYNQ-Z2 の Raspberry Pi 40 ピンヘッダと Arduino ヘッダは SLMLET チップの IO と接続されており，プログラムのブートやリセット，SPI によるデータ転送などを行う．図 6 に示すように，PYNQ-Z2 の Programmable Logic(PL) 部にはこれらの制御を行うハードウェアが実装されており，PySLMLET と呼ぶドライバソフトウェアを用いて Python で記述されたソフトウェアから容易に扱うことができる [5]．

本研究では，PySLMLET を拡張し，新たに Text-based User Interface (TUI) アプリケーション (pyslmlet-tui) を用意した．コマンドラインから pyslmlet-tui を実行するコマンドと共に，3. 節のフローで作成された実行バイナリやその他動作条件 (動作周波数や電圧など) のオプションを与えることで，指定されたプログラムのテストを行うことができる．初期実装では，pyslmlet-tui を実行するたびに，PYNQ-Z2 の PL 部をコンフィギュレーションするのに必要な *pynq* パッケージのインポートが必要があった，しかし，PYNQ-Z2 に搭載されるプロセッサ (ARM Cortex-A9) ではこれに 20 秒近くの時間を要することが判明した．一方で，PL 部に構成する回路は SLMLET でプログラムを実行するたびに再構成し直す必要はなく，一度だけ再構成すれば良い．そこで，PL 部を再構成し，PL 部のハードウェアと通信する部分をサーバープログラムとして分離し，SLMLET でプログラムを実行する際は，クライアントプログラムがサーバーに接続するように設計した．このような方式にしたことで，pyslmlet-tui は PYNQ-Z2 ボードだけでなく，これと同一ネットワーク上に存在する端末でも実行可能になった．例えば，AMD Ryzen 7 5700G を搭載する汎用の Linux PC で pyslmlet-tui を実行すると，コマンド実行から 1 秒もかからずに SLMLET でプログラムを開始できるようになった．

pyslmlet-tui には加えて VISA/SCPI に対応した直流電源を操作する機能も備わっており，簡単なシェルスクリプトを用意することでさまざまな測定条件での実行と評価結果の取得までをすべて自動で実行できるようになった．

5. 評　　価

5.1 SLM 部の再構成

はじめに，実装したライブラリおよび測定環境を用いて SLM 部およびコンフィギュレーションコントローラの動作特性を測定した．SLM 部のコンフィギュレーションに用いる回路として RISC-V コアから読み書き可能なメモリマップトな 32 ビットレジスタ 4 本 (以降 MMReg と呼ぶ) を VerilogHDL で設計し，これを評価に用いる．

図 7 に 50MHz のシステムクロックを入力した際に，SLM の 2 つのブロックを同じ MMReg で再構成するのに要する時間を計測した．計測には Cadence NC-Verilog による RTL シミュレーションを用いている．計測区間は `loadBitstream` で共有 SRAM 上にビットストリームをロードし，`configurationSLM` または `configurationSLMAsync` によりビットストリームを

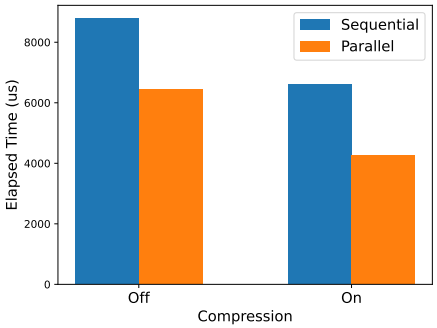
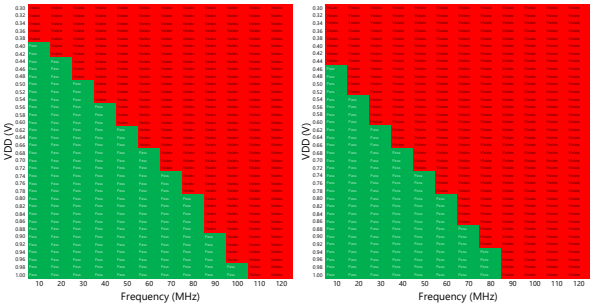
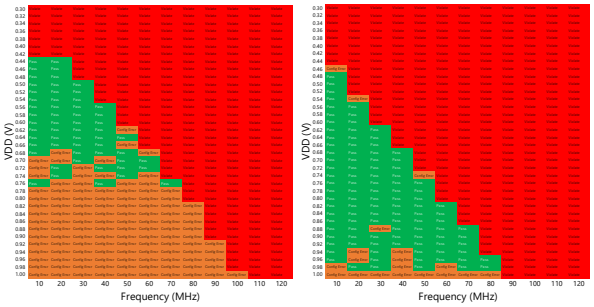


図 7: SLM2 ブロックを再構成するのに要する時間@50MHz



(a) ブロック 1 (VPW = 0.0 V)　(b) ブロック 1 (VPW = -0.4 V)



(c) ブロック 2 (VPW = 0.0 V)　(d) ブロック 2 (VPW = -0.4 V)

図 8: SLM ブロックおよびコンフィギュレーションコントローラの動作範囲

SLM ブロックに書き込みが完了するまでとした．さらに，TLC による圧縮を有効にした場合としない場合，2 つの SLM ブロックを並列に再構成する場合としない場合をそれぞれ組み合わせて計測を行った．MMReg の BLE 使用率は 29.3% であり，圧縮後のビットストリームは非圧縮時と比べ 49.8 である．3.1 節で述べた通り，SLM ブロックに書き込むビットストリームデータ量は圧縮によらず固定であるが，`loadBitstream` に要する時間が短縮できるため，非圧縮時と比較して 24.8%(逐次)，33.9%(並列) の時間短縮ができている．また，並列再構成を行う場合，SRAM へのビットストリーム展開は RISC-V コアで行うため，並列化できない．そのため，実行時間は半分とはならないものの，逐次実行の場合と比べ非圧縮時では 26.6%，圧縮時で 35.4% の時間短縮となる．

先行研究の CIFER[1] は 6720 LUT-6 の FPGA を再構成するのに 1ms 程度の時間である．一方で，CIFER の CPU は本評価の 25 倍近い周波数で動作可能であることとを考慮す SLMLET の再構成時間は十分に短いと言える．

次に，4. 節で述べた評価環境を用いて実機の動作範囲を確認する．DDC プロセスの標準電圧は 0.90V であるが，0.30-1.00V の範囲を 0.02V 刻みで動作テストを行なった．また，nMOS 側のボディバイアス電圧 VPW を標準状態 0.0V(ゼロバイアス) から-0.4V(リバースバイアス) にした状態で同様のテストを行

表 1: 評価結果の比較

			アプリケーション		
			sram_memcpy	CRC32	AES128
SLMLET (soft)		f_{max}	300 MHz	300 MHz	300 MHz
		Cycle	**28,679**	14,117	1,865
		P_{chip}	428.6 mW	393.3 mW	426.8 mW
		Latency	95.60 us	47.05 us	**6.217** us
		Energy	40.98 uJ	18.50 uJ	2.653 uJ
SLMLET		f_{max}	100	70	28
		Cycle	32,994	**2,151**	**976**
		P_{chip}	268.5 mW	170.0 mW	158.9 mW
		Latency	329.9 us	**30.73** us	34.86 us
		Energy	88.60 uJ	**5.22** uJ	6.518 uJ
	Resource	Logic(SLM)	42.2%	42.1%	87.5%, 94.2%
		FF	18.3%	14.6%	32.7%, 15.5%
		LB	45%	51%	94%, 98%
ESP32[a]		P_{total}	n/a	312.5 mW	306.0 mW
		Latency	n/a	77.58 us	7.814 us
		Energy	n/a	24.24 uJ	2.391 uJ
RP2040[b]		P_{total}	n/a	154.8 mW	158.9 mW
		Latency	n/a	66.59 us	14.00 us
		Energy	n/a	10.31 uJ	**2.223** uJ
Tang Nano 9K		f_{max}	n/a	50 MHz	34 MHz
		P_{comp}	n/a	281.9 mW	618.5 mW
		P_{trans}	n/a	268.3 mW	278.7 mW
		Latency	n/a	885.7 us	1333 us
		Energy	n/a	335.0 uJ	237.9 uJ
	Resource	Logic	n/a	22.9 %	39.7 %
		FF	n/a	13.6%	20.0%
		CLS	n/a	32.9%	57.2%
		BSRAM	n/a	12%	8.0 %

[a] 使用可能な最高周波数 240MHz で計測

[b] 200MHz で計測

なった. リバースバイアスはトランジスタの動作速度が低下, すなわち動作可能な周波数が低下するが, リーク電力を削減することができる状態である.

図 8 に, 動作範囲をプロットした結果を示す. 赤い部分は動作に失敗した条件で, 緑の部分が成功した条件である. ブロック 1 の場合 (図 8(a)) を見ると, 10MHz であれば 0.40V 程度で正しく動作し, 1.0V 程度まで電圧を上げることで 100MHz まで動作可能であることがわかった. また, -0.4V のリバースバイアス (図 8(b)) の場合, 動作範囲が 80MHz まで低下し, またゼロバイアス時と比べ各周波数で動作に必要な最低電圧が高くなっている. これらの傾向は予想通りのものであり, 期待通りの結果が確認できた.

一方で, ブロック 2 に関しては, 高い電圧を印加すると 10MHz であっても正しく動作しないという問題が明らかになった. 解析の結果, コンフィギュレーションコントローラによる再構成自体が正しく行われていないことがわかった. そこで, SLM ブロックの再構成までを正しく動作が確認できた条件で行い, 完了後テスト条件の電圧, 周波数に変更して動作テストを行なった. その結果が図 8(c) と 8(d) である. 図中のオレンジの部分はこの条件で再構成には失敗するものの, 正しく再構成されたのちにこの条件にした場合 SLM ブロック上の回路は正しく動作したケースを表す. SLM ブロック上の回路の動作範囲はブロック 1 の場合と大きな違いがないことがわかる. また, リバースバイアスにすることで, 再構成が失敗する範囲が減るという現象を確認した. 今後はさらに詳細な解析を行い, この動作不良の原因を究明する.

5.2 ケーススタディ

本節ではケーススタディとして 3 つのベンチマークアプリケーションを用いて試作した SLMLET チップの評価を行う. ベンチマークは以下の通りである:

- sram_memcpy: 共有 SRAM 上の 16KB データをコピー
- CRC32: 1KB のバイナリデータから CRC32(生成多項式 0x04C11DB7) を計算する
- AES128: 128bit 長の AES 暗号化を 1 ブロック計算する

各アプリケーションは, SLM 部のハードウェアで実行されるバージョンと, すべてを RISC-V コアで実行されるソフトウェア実装版を用意した. CRC32 のハードウェアは 1 サイクルで 32bit の CRC を計算する設計である. また, AES128 は一つの SLM ブロックではリソースが不足していたため, 2 つのブロックを利用している. ソフトウェア実装に関しては CRC32 と AES128 ともに MiBench の実装を利用した.

sram_memcpy 以外のアプリケーションについては SLMLET の他にエッジ,IoT 向けのマイクロコントローラである Espressif Systems 社の ESP32 および RaspberryPi Pico(RP2040) においても各種測定を行なった. 同一のソフトウェア記述をそれぞれ ESP IDF 5.0.2 および Pico SDK 1.5.1 を用いてプログラムを作成した. 実行時間の時間計測については, 計測区間を 10000 ループさせ 1 回あたりの時間を得た. ソフトウェアのコンパイルについてはどのシステムにおいても O3 オプションを用いた.

さらに, ディスクリートな FPGA として Gowin GW1NR-9 を搭載する TangNano 9K を選び, CRC32 と AES128 を計算するハードウェアを実装し, 評価を行なった. FPGA の設計は Gowin 1.9.8.11 IDE Education version を用いた. データの入出力には UART を介して行い, 提供される UART MASTER IP を用いた. ただし, IP 生成時に指定するボーレートは正しくデータ転送が確認できた最大値の 921600 を指定している. また, UART IP と演算処理部のハードウェアでクロックドメインの分割を行うために, ドメイン境界では 4-phase handshake を行う.

表 1 に評価結果をまとめる. SLMLET についてはシミュレーションにより取得した計測区間のサイクル数や 5.1 節と同様の電圧条件のもとで最大の動作周波数 f_{max} とその時のチップが消費する電力 P_{chip}, 処理レイテンシ, 消費エネルギーを示している. また, SLM ブロックを用いる場合は, 各回路をのせた場合のリソース使用率も合わせて記載している.

RISC-V のコア部は, アプリケーションによらず 300MHz での動作に成功した. 標準電圧である 0.90V 印加時でも 280MHz での動作が確認できている. sram_memcpy に関しては, 3.1 節で述べた通り SLM ブロックは 16bit 幅でしか共有 SRAM にアクセスできないため, ソフトウェア実装と比較して所用サイクル数が多くなっている. 加えて, SLM ブロック上の回路は最大で 100MHz でしか動作できないため, f_{max} での動作時では, レイテンシは 3 倍ほどである. 結果, ソフトウェア実装の方が SLM ブロックを利用する場合よりもおよそ半分程度の消費エネルギーとなる. ただし, SLM ブロックを利用する場合, SRAM 上のデータをコピーしている間に, コア部やもう片方の SLM ブロックが別の処理を行うことは可能である.

AES128 については, 2 つの SLM ブロックを跨ぐ経路がクリティカルパスとなっていると考えられ, 他のアプリケーションと比べ f_{max} が 28MHz と低い. そのため, 所用サイクル数はソフトウェア実装と比べ半分程度であるが f_{max} が 10 分の 1 以下であるため, レイテンシはソフトウェア実装の方が小さい. 同

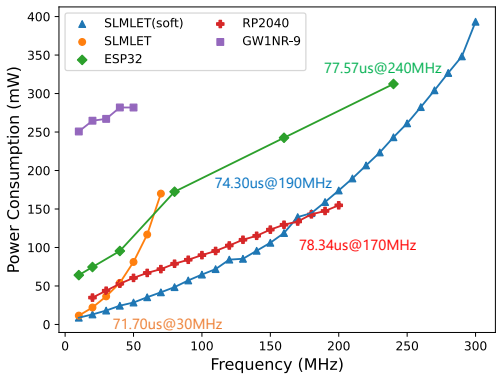


図 9: CRC32 消費電力の比較

様に, ESP32 や RP2040 と比べても SLM ブロックを利用したケースの方がレイテンシが大きく, 消費エネルギーも高いという結果となった. ただし, 30MHz 程度の動作周波数でこれらを比較すれば, 各ソフトウェア実装は 50-90us 程度のレイテンシとなるため, SLM ブロックを使用した方が高速である. この周波数ではソフトウェア実装の 3 種類で比較しても SLMLET がレイテンシは最も小さいものの, 必要な動作電圧が高いためその分消費エネルギーのペナルティが高い. したがって, 消費エネルギーはいずれのソフトウェア処理も優れる.

CRC32 に関しては, SLM ブロックを利用することで所用サイクルを 85% ほど削減することができる. そのため, SLM 上の CRC32 用回路が 70MHz までの動作であったとしても, レイテンシは SLM ブロックを利用した方が短く, 消費エネルギーをおよそ 70% ほど削減することができる. また, レイテンシは ESP32 と比べて 39%, RP2040 と比べて 46% と短く, 消費エネルギーもそれぞれと比較して 21%, 50% と大きく削減できている.

さらに, 図 9 に CRC32 実行時の各周波数ごとの消費電力を示す. SLM ブロック利用時の SLMLET で表 1 に示した 70MHz 動作時は電源電圧を 1.0V 近くまで上げる必要があったため, 同一の周波数では他のデバイスよりも電力が高い. しかし, 図中にプロットしている通り各デバイスにおいてレイテンシがおおよそ 70us となる周波数で見れば, SLMLET は 30MHz で動作すればよく, この時に電力が最小となるのは電源電圧 0.64V とボディバイアス電圧 (VPW)-0.3V のリバースバイアス状態にした時である. この時, リーク電力とダイナミック電力を共に大きく削減することができる. ゆえに, 消費電力も 36mW 程度と小さい. 一方で, 各デバイスでソフトウェア実行した場合の同等のレイテンシとなる周波数では, SLMLET で 159mW, ESP32 で 312mW, RP2040 で 133mW と非常に大きい. このように, ソフトウェア実行と比べ大きく所要サイクル数が小さくなるアプリケーションにおいては, 動作周波数を下げ電源電圧, ボディバイアス電圧を最適化することで大きなエネルギー削減が期待できる.

最後に, TangNano 9K との比較を行う. プロセッサと密結合していないため, UART によるデータの入出力が実行時間の大半を占める. ただし, CRC32 の処理データ 1KB を転送する時間は計測区間に含まれていない. さらに, CRC32 のようにリソース使用量が多くないアプリケーションにおいてはダイナミック電力の割合が小さく, スタンバイ状態でも 250mW 程度の電力を消費していた. 以上の理由からどちらのアプリケーションにおいても他のデバイスと比べ 1 桁から 2 桁以上大きい消費エネルギーとなった. TangNano 9K の GW1NR-9 は 55nm プロセスで実装されており, プロセスサイズは SLMLET の試作チップと同じである. CRC32 については, 演算処理部の回路設計が完全に同じであり, 動作周波数も近い. しかし, SLMLET で 50MHz 時には-0.3V 程度のリバースバイアスを用いる余裕があり, この時のリーク電力は 5mW である. SLMLET の FPGA のリソース規模が GW1NR-9 と比べ 9 分の 1 程度であることを考慮しても, リーク電力は極めて小さく, FPGA などのチップ面積が大きくなるシステムにおいて DDC プロセスを使用する効果は大きい.

6. おわりに

本稿では RISC-V コアと SLM 再構成ロジックを構成要素として持つ SLMLET 向けに開発したアプリケーション開発フローおよびライブラリについて紹介した. これらを用いて実機測定を行ったところ, アプリケーション特化なハードウェアを SLM ブロック上に実装し, プロセッサの代わりに処理をさせることで, 消費エネルギーを最大で 80% ほど削減できることを示した. さらに, プロセッサと SLM ブロック間のデータ転送オーバーヘッドがディスクリート型の FPGA と比べ小さいため, オフローディングが有効なアプリケーションの適用範囲が広がると推測される. また, ボディバイアス制御などの最適化により消費エネルギーをさらに改善できる可能性が示唆され, 今後は FPGA CAD のタイミングレポートなどから最適な電圧を決定する手法などを検討していく.

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.

文　　献

[1] T.-J. Chang, A. Li, F. Gao, T. Ta, G. Tziantzioulis, Y. Ou, M. Wang, J. Tu, K. Xu, P.J. Jackson, et al., “CIFER: A 12nm, 16mm 2, 22-Core SoC with a 1541 LUT6/mm 2 1.92 MOPS/LUT, Fully Synthesizable, CacheCoherent, Embedded FPGA,” 2023 IEEE Custom Integrated Circuits Conference (CICC)IEEE, pp.1–2 2023.

[2] M. Kuga, Q. Zhao, Y. Nakazato, M. Amagasaki, and M. Iida, “An eFPGA Generation Suite with Customizable Architecture and IDE,” IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, vol.106, no.3, pp.560–574, 2023.

[3] K. Fujita, Y. Torii, M. Hori, J. Oh, L. Shifren, P. Ranade, M. Nakagawa, K. Okabe, T. Miyake, K. Ohkoshi, et al., “Advanced channel engineering achieving aggressive reduction of V T variation for ultra-low-power applications,” 2011 International Electron Devices MeetingIEEE, pp.32–3 2011.

[4] Q. Zhao, M. Amagasaki, M. Iida, M. Kuga, and T. Sueyoshi, “An automatic FPGA design and implementation framework,” 2013 23rd International Conference on Field programmable Logic and ApplicationsIEEE, pp.1–4 2013.

[5] 小島拓也, 亀井愛佳, 矢内洋祐, 天野英晴, 久我守弘, 飯田全広他, “Jupyter Notebook を介した RISC-V SoC 向け実機テスト環境の構築,” 研究報告組込みシステム (EMB), vol.2023, no.24, pp.1–7, 2023.

[6] S. Takagi, N. Niwa, Y. Yanai, H. Amano, M. Amagasaki, Y. Nakazato, and M. Iida, “Tag-less compression for FPGA configuration data,” *Proc. of SASIMI 2022*, pp.81–82, 2022.