Algorithmic Complexity vs Data Width: A Comparative Analysis of 16-bit and 32-bit MIPS Processor Implementations on FPGA

Ricardo Herrera-Pineda
ECE Department, Computer Engineering
California State Polytechnic University, Pomona
Pomona, California, USA
rpineda1@cpp.edu

Sean Wygant
ECE Department, Computer Engineering
California State Polytechnic University, Pomona
Pomona, California, USA
smwygant@cpp.edu

Daniel Mondragon
ECE Department, Computer Engineering
California State Polytechnic University, Pomona
Pomona, California, USA
dxicotencatl@cpp.edu

Heba Hafez
ECE Department, Computer Engineering
California State Polytechnic University, Pomona
Pomona, California, USA
hhafez@cpp.edu

***Abstract*—This paper presents a comparative analysis of 16-bit and 32-bit single-cycle MIPS processor implementations on FPGA hardware, revealing counterintuitive findings about the relationship between data path width and resource utilization. Conventional wisdom suggests that narrower data paths result in more efficient hardware implementations. However, empirical evaluation demonstrates that the 16-bit processor consumes 10% more lookup tables (LUTs) than its 32-bit counterpart (414 vs 376 LUTs) while executing 21% more instructions for identical computational tasks. Through detailed analysis of synthesized netlists and performance metrics, inefficient branch calculation logic is identified as the primary cause of this paradox. The 16-bit implementation employs a six-operation branch algorithm with manual two's complement handling, while the 32-bit version utilizes a two-operation approach that trusts signed arithmetic. The findings establish that algorithmic efficiency dominates resource utilization more than data width, with implications for educational processor design and hardware optimization strategies. Performance measurements confirm the 32-bit processor achieves 21% faster execution while consuming only 4% more power.**

***Index Terms*—MIPS processor, FPGA implementation, hardware optimization, two's complement arithmetic, resource utilization, algorithmic complexity**

I. Introduction

Processor architecture design involves fundamental trade-offs between data path width, computational complexity, and hardware resource utilization. Educational processor implementations typically prioritize simplicity and resource efficiency, leading to the common practice of implementing narrow data paths to minimize hardware requirements [1]. The MIPS (Microprocessor without Interlocked Pipeline Stages) architecture serves as a canonical example for teaching computer architecture principles due to its simplified instruction set and clear datapath organization [2].

This study challenges the assumption that narrower data paths inherently produce more efficient implementations. While a 16-bit datapath handles half the data width of a 32-bit implementation, the relationship between data width and total resource consumption depends critically on algorithmic design choices. Previous work focuses primarily on pipelined architectures and their associated hazards [3], leaving single-cycle implementations and their resource trade-offs underexplored in the context of FPGA synthesis.

The research addresses four key questions:

RQ1: Does data path width directly correlate with hardware resource utilization in single-cycle MIPS implementations?

RQ2: How do algorithmic design choices impact resource consumption independent of data width?

RQ3: What are the performance implications of different branch calculation strategies in MIPS processors?

RQ4: Can these findings inform future educational processor implementations and optimization strategies?

This paper presents empirical evidence from two complete processor implementations synthesized on a Nexys A7-100T FPGA platform, demonstrating that algorithmic efficiency—specifically in branch address calculation—dominates resource utilization patterns more than data width considerations alone.

II. Contribution

This work provides three primary contributions to the field of educational processor design and hardware optimization:

A. Empirical Demonstration of Algorithmic Complexity Dominance

The study quantifies the counterintuitive finding that a 16-bit processor can consume more logic resources than a 32-bit implementation, establishing that algorithmic complexity contributes more significantly to resource utilization than data width. This principle has immediate implications for educational curricula and hardware design courses.

B. Detailed Analysis of Branch Calculation Inefficiency

Through systematic examination of synthesized netlists and Verilog implementations, the research identifies and characterizes a specific algorithmic inefficiency: manual two's complement handling in branch calculations. The 16-bit implementation employs six distinct operations (shift, negate, increment, compare, add, subtract, multiplex) while the 32-bit version accomplishes the same result with two operations (shift, add). This analysis provides concrete evidence of how trusting hardware features (signed arithmetic) produces more efficient implementations than manually reimplementing those features.

C. Comprehensive Performance Characterization

The study presents a complete evaluation spanning logic utilization (LUTs), storage utilization (flip-flops), power consumption, timing characteristics, and execution performance. This multi-dimensional analysis reveals consistent patterns across all metrics, establishing clear relationships between algorithmic choices and their hardware manifestations.

III. Related Work

A. Educational Processor Implementations

Harris and Harris [1] present standard MIPS implementations focusing on pipelined architectures with hazard detection and forwarding units. Their work emphasizes the complexity introduced by pipelining but does not systematically compare single-cycle implementations of different data widths. Patterson and Hennessy [2] establish foundational principles for MIPS architecture but primarily focus on 32-bit implementations without exploring the resource implications of narrower designs.

B. FPGA-Based Processor Studies

Several researchers explore MIPS implementations on FPGA platforms. Kumar et al. [4] investigate pipelined MIPS processors on Xilinx FPGAs, focusing on clock frequency optimization. Their work achieves high performance but does not address the fundamental question of how data width and algorithmic complexity interact. Chen and Wang [5] present a reduced MIPS implementation for educational purposes but do not provide detailed resource utilization comparisons.

C. Hardware Optimization Research

Existing research on hardware optimization typically focuses on power consumption [6] or area minimization through techniques such as clock gating, resource sharing, and voltage scaling. However, these studies generally assume that algorithmic implementations are fixed and focus on lower-level optimizations. This work operates at a higher level of abstraction, demonstrating that algorithmic choices fundamentally determine resource requirements before lower-level optimizations can be applied.

D. Gap in Literature

The literature lacks systematic comparisons of how algorithmic design choices—specifically in fundamental operations like branch calculation—affect resource utilization across different data widths in single-cycle processors. This study fills that gap by providing empirical evidence and detailed analysis of the mechanisms through which algorithmic complexity manifests in hardware resources.

IV. System Architecture

A. Implementation Overview

Both processors implement single-cycle MIPS architectures synthesized for the Nexys A7-100T FPGA (Xilinx Artix-7, xc7a100tcsg324-1). The designs share fundamental components but differ in data path width and branch calculation algorithms. Table I summarizes the architectural specifications.

TABLE I
Architectural Specifications

Component	16-bit	32-bit
Data path width	16 bits	32 bits
Register count	8	32
Register width	16 bits	32 bits
Instruction width	16 bits	32 bits
Instruction formats	R, I, J	R, I, J
Operations	13	13
Max memory	64 KB	4 GB
PC increment	+2 bytes	+4 bytes

B. Common Components

Both implementations include standard MIPS components: Program Counter (PC) tracking current instruction address; Instruction Memory storing the program; Register File with register $0 hardwired to zero; Arithmetic Logic Unit (ALU) performing operations (ADD, SUB, AND, OR, SLT); Data Memory for load/store operations; and Control Unit decoding instructions and generating control signals.

C. Critical Difference: Branch Calculation

The fundamental architectural divergence occurs in branch address calculation.

16-bit Branch Implementation:

```
assign im_shift_1 = {imm_ext[14:0], 1'b0};
assign no_sign_ext = ~(im_shift_1) + 1'b1;
assign PC_beq = (im_shift_1[15] == 1'b1) ?
    (pc2 - no_sign_ext) : (pc2 + im_shift_1);
```

This implementation performs six operations: shift, bitwise NOT, increment, sign comparison, subtraction, and addition, with multiplexer selection.

32-bit Branch Implementation:

```
assign im_shift_2 = {imm_ext[29:0], 2'b00};
assign PC_beq = pc_plus_4 + im_shift_2;
```

This implementation performs two operations: shift and addition.

The 16-bit approach stems from misunderstanding two's complement arithmetic. The code manually checks the sign bit and uses subtraction for negative offsets, not recognizing that addition automatically handles signed operands. The 32-bit implementation trusts the adder to process signed numbers, leveraging the property that adding a negative number produces the same result as subtraction.

D. Test Program

Both processors execute an identical summation algorithm computing $0 + 1 + 2 + ... + 49 = 1,225$. The program uses a loop with counter initialization, comparison, accumulation, increment, and branching. The accumulator register value displays on FPGA LEDs, providing visual confirmation of correct execution.

V. Experimental Methodology

A. Development Environment

Hardware: Nexys A7-100T FPGA (Xilinx Artix-7, 63,400 LUTs, 126,800 FFs, 100 MHz clock divided to 10 kHz)

Software: AMD Xilinx Vivado Design Suite 2023.2, Verilog HDL

B. Measurement Procedures

Resource Utilization: Vivado post-implementation reports for LUTs, flip-flops, and I/O blocks.

Power Analysis: Vivado Power Estimator provides dynamic and static power breakdown.

Timing Analysis: Vivado Timing Analyzer reports endpoints and timing paths.

Performance: Execution time measured via video recording; instruction count from simulation.

Power Measurement: USB power meter (ChargerLAB POWER-Z) measures actual consumption.

C. Experimental Controls

Both processors use identical synthesis settings, clock frequencies, test programs, external loads (16 LEDs), FPGA chip, board, and environmental conditions.

VI. Results

A. Resource Utilization

Table II presents synthesis and implementation results.

TABLE II
FPGA Resource Utilization

Resource	**16-bit**	**32-bit**	**Diff**	**%**
LUTs	414	376	+38	+10.1%
Flip-Flops	116	196	+80	+69.0%
I/O Blocks	21	21	0	0%

The results reveal a counterintuitive pattern: despite half the data width, the 16-bit processor consumes 10% more LUTs. Netlist analysis attributes this to branch calculation logic requiring approximately 90 LUTs (16-bit) versus 40 LUTs (32-bit), a difference of 50 LUTs that exceeds the total measured difference.

Flip-flop utilization follows expected patterns: wider data paths require more storage. The 69% increase aligns with theoretical predictions based on PC width, register file size, and data path components.

B. Power Consumption

Table III presents power analysis results.

TABLE III
Power Consumption Analysis

Component	**16-bit**	**32-bit**
Vivado Estimates:		
Total On-Chip	6.604 W	6.613 W
Dynamic Power	6.466 W	6.475 W
I/O	6.164 W	6.164 W
Signals	0.176 W	0.190 W
Logic	0.126 W	0.120 W
Device Static	0.138 W	0.138 W
USB Meter:		
Actual Power	0.833 W	0.862 W

Key findings: (1) I/O dominates at 93%, dwarfing processor differences; (2) 32-bit uses less logic power (-4.8%) but more signal power (+8.0%), confirming simpler algorithm reduces computational power while wider paths increase interconnect power; (3) Vivado overestimates due to theoretical maximum I/O calculations.

C. Timing Analysis

Table IV presents timing characteristics.

TABLE IV
Timing Characteristics

Timing Check	**16-bit**	**32-bit**	**Diff**
Internal Endpoints	887	551	+336 (+61%)
No Clock	180	196	-16 (-9%)
No Output Delay	16	16	0
No Input Delay	1	1	0

The 61% increase in internal endpoints reveals timing complexity from inefficient branch logic. Endpoints represent flip-flops receiving signals; complex logic with intermediate calculations requires more temporary storage.

D. Execution Performance

Table V summarizes execution performance.

TABLE V
Execution Performance

Metric	**16-bit**	**32-bit**	**Diff**
Instructions	20,354	16,836	+21%
Time (10 kHz)	8.07 s	6.70 s	+21%
CPI	1.0	1.0	0
Result	1,225	1,225	0

Both processors correctly compute the sum (1,225). The 16-bit processor executes 21% more instructions due to instruction encoding differences and compressed format limitations. Execution time directly reflects instruction count (single-cycle, CPI = 1.0).

VII. Discussion

A. *Algorithmic Complexity Dominance (RQ1 & RQ2)*

The results definitively answer RQ1 and RQ2: data width does not directly correlate with resource utilization; algorithmic design choices dominate. Evidence includes 10% higher LUT usage, branch logic contribution of 50 LUTs, logic power following LUT count, and 61% higher endpoint count for the 16-bit implementation.

The branch calculation serves as a case study. The 16-bit manual two's complement handling creates separate adder/subtractor circuits, NOT gate array (16 LUTs), incrementer (16 LUTs), sign comparator (2 LUTs), and multiplexer (16 LUTs). The 32-bit implementation eliminates this overhead with a single adder.

B. *Two's Complement Arithmetic Principle*

The fundamental insight lies in two's complement properties. Addition of a negative number automatically produces correct results without sign checking. The 16-bit code unnecessarily detects negative offsets, negates to positive, then subtracts—but the subtractor internally performs the equivalent of adding the negative value, resulting in double negation.

C. *Performance Implications (RQ3)*

Performance analysis (RQ3) reveals multiple impacts: 21% faster execution, better energy efficiency (14% less total energy), and better scalability to higher frequencies due to simpler critical paths.

D. *Educational Implications (RQ4)*

Findings inform educational implementations (RQ4): provide concrete examples of algorithmic efficiency importance, establish design guidelines (trust hardware features, prefer simple algorithms, use single arithmetic paths), and offer rich curriculum material demonstrating how design choices impact multiple metrics.

VIII. Conclusion

This study demonstrates that algorithmic efficiency dominates hardware resource utilization more than data path width in processor implementations. The 16-bit MIPS processor consumes 10% more lookup tables than the 32-bit version due to inefficient branch calculation logic that manually reimplements two's complement arithmetic instead of trusting the adder's signed arithmetic capabilities.

Key findings establish: (1) algorithmic complexity determines resource consumption; (2) resource metrics show consistent patterns favoring simpler algorithms; (3) performance improvements compound (21% faster, 14% less energy); (4) educational value in demonstrating hardware optimization principles.

Future educational processor implementations should prioritize algorithmic simplicity and leverage built-in hardware features rather than assuming narrower data paths automatically produce more efficient designs. The research establishes that smart design beats brute-force narrowing—a principle with broad applicability beyond educational contexts.

References

[1] S. Harris and D. Harris, *Digital Design and Computer Architecture*, 2nd ed. Morgan Kaufmann, 2012.

[2] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design: The Hardware/Software Interface*, 5th ed. Morgan Kaufmann, 2014.

[3] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Morgan Kaufmann, 2017.

[4] A. Kumar, P. Singh, and R. Sharma, "FPGA implementation of pipelined MIPS processor with hazard detection," *International Journal of Computer Applications*, vol. 93, no. 6, 2014.

[5] Y. Chen and L. Wang, "Design and implementation of a simplified MIPS processor for teaching purposes," in *Proc. International Conference on Engineering Education*, 2015.

[6] M. Pedram and J. M. Rabaey, *Power Aware Design Methodologies*. Springer, 2002.