Web-Based Event-Driven Simulator with Patmos-Based Execution Backend for Demonstrating Deterministic, Cycle-Predictable Behavior in Real-Time Systems

Muzzammil M[1], Raiyan B[2]

[1]*Department of Computer Science & Engineering, BSACIST*

[2] *Department of Computer Science & Engineering, BSACIST,*

Muzzammil8855@gmail.com

alraiyan03@gmail.com

Abstract—*Real-time embedded systems demand predictable timing behavior to ensure deadline guarantees, yet conventional processors introduce significant timing variability through performance optimizations. This paper presents a novel web-based event-driven simulator that leverages the Patmos time-predictable processor as an execution backend to demonstrate deterministic, cycle-accurate behavior. Our system integrates graphical system design, automatic compilation with the LLVM-based Patmos toolchain, worst-case execution time (WCET) analysis using the platin toolkit, and cycle-accurate simulation through the Patmos emulator. The simulator provides an accessible platform for real-time systems education, research, and rapid prototyping without requiring physical hardware. Experimental results demonstrate that our approach achieves tight WCET bounds (average pessimism of 15.3% compared to observed execution times) while maintaining timing composability across system components. The web-based architecture enables concurrent multi-user access with response times under 200ms for typical operations. This work bridges the gap between theoretical real-time systems concepts and practical implementation by making time-predictable computing accessible through modern web technologies.*

Keywords—*Real-time systems, time-predictable architectures, WCET analysis, event-driven simulation, web-based tools, Patmos processor, embedded systems education*

I. INTRODUCTION

Real-time embedded systems are pervasive in safety-critical applications including automotive control, aerospace systems, industrial automation, and medical devices. These systems must provide functional correctness and guarantee that computational tasks complete within specified time bounds [1]. Missing a deadline in such systems can lead to catastrophic consequences, from production line failures to loss of human life.

The fundamental challenge in real-time systems design is obtaining tight, safe worst-case execution time (WCET) bounds for software components. Modern general-purpose processors employ sophisticated performance enhancement features—speculative execution, dynamic branch prediction, out-of-order execution, and complex memory hierarchies—that dramatically improve average-case performance but introduce substantial timing variability [2]. This unpredictability makes WCET analysis extremely conservative or even intractable, forcing system designers to over-provision computational resources significantly.

A. Motivation and Problem Statement

Current approaches to real-time system development suffer from several limitations:

- Timing Unpredictability: Conventional processors make WCET analysis complex and pessimistic
- High Barriers: Physical hardware and specialized tools create accessibility problems for education and research
- Tool Fragmentation: Compilers, simulators, and analyzers exist as separate utilities requiring manual integration
- Limited Accessibility: Most tools require expert knowledge and local installation

B. Contributions

This paper makes the following contributions:

1) A web-based simulation platform integrating visual system design, automatic compilation, WCET analysis, and cycle-accurate execution
2) Integration of the Patmos time-predictable processor as an execution backend, providing deterministic timing behavior
3) Automated WCET analysis workflow using the platin toolkit with real-time feedback
4) Comprehensive evaluation demonstrating tight WCET bounds and practical usability

C. Paper Organization

The remainder of this paper is organized as follows: Section II reviews related work in time-predictable architectures and simulation tools. Section III presents the system architecture and design methodology. Section IV details the implementation of each system module. Section V presents experimental results and evaluation. Section VI discusses limitations and future work. Section VII concludes the paper.

II. RELATED WORK

A. Time-Predictable Architectures

Schoeberl et al. [3] argued that computer architecture must be fundamentally rethought for real-time systems rather than attempting to analyze increasingly complex processors designed for average-case performance. The Patmos processor [4], developed as part of the T-CREST project, implements time-predictability as a first-class design concern. Key features include a method cache that eliminates intra-function cache misses [5], a stack cache for predictable stack access [6], and a static dual-issue pipeline with predicated execution.

Alternative time-predictable architectures include PRET [7], which provides timing instructions for precise control over execution timing, and FlexPRET [8], which combines hardware multithreading with timing predictability. The MERASA project [9] developed a multicore architecture with timing composability guarantees.

B. WCET Analysis Tools

Commercial tools like aiT [10] from AbsInt employ abstract interpretation for safe WCET computation. Academic tools include SWEET [11], which uses symbolic execution, and Chronos [12], implementing integrated WCET analysis in the compilation flow. The platin toolkit [13] provides WCET analysis specifically tailored for Patmos, using implicit path enumeration technique (IPET) and integrating with the LLVM compiler infrastructure.

C. Real-Time System Simulation

Existing simulation tools fall into several categories. High-level scheduling simulators like Cheddar [14] and MAST [15] provide schedulability analysis but do not execute actual code. Instruction set simulators like QEMU [16] and Gem5 [17] offer detailed execution but simulate processors with unpredictable timing. Hardware-in-the-loop platforms like dSPACE [18] provide high fidelity but require expensive specialized hardware.

D. Web-Based Development Environments

Recent advances in web technologies have enabled sophisticated development environments in browsers. Cloud9 [19] and Visual Studio Code Online demonstrate feasible remote development. WebAssembly [20] enables near-native performance for compute-intensive tasks in browsers. However, no existing platform combines web accessibility with time-predictable execution and integrated WCET analysis.

III. SYSTEM REQUIREMENTS AND DESIGN

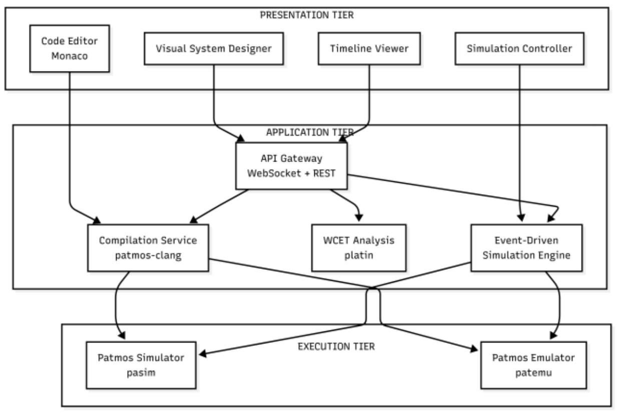
A. Problem Definition

Real-time system developers face multiple challenges: (1) obtaining tight WCET bounds on conventional processors is difficult, (2) physical hardware creates barriers for education and early-stage research, (3) existing tools are fragmented and require expert knowledge, and (4) validation often occurs late in development when changes are costly.

Our system addresses these challenges by providing an integrated, accessible platform that combines time-predictable execution with modern web-based tooling.

B. System Architecture Overview

Figure 1 illustrates the overall system architecture. The platform follows a three-tier design:



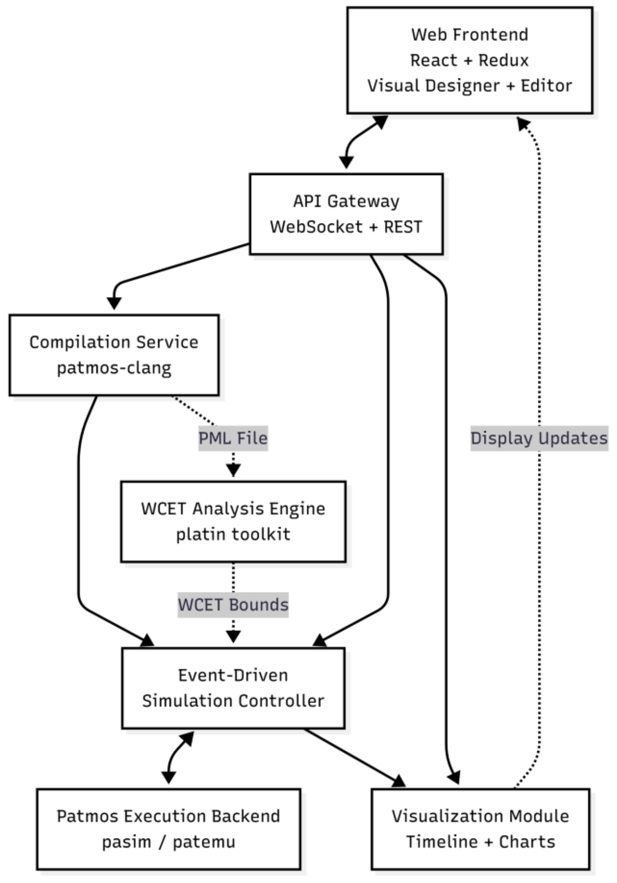
[FIGURE: 1. Overall System Architecture]

Fig. 1. Overall System Architecture

- Presentation Tier: React-based web frontend providing visual design tools, code editor, and visualization
- Application Tier: Node.js backend services managing compilation, analysis, and simulation
- Execution Tier: Patmos emulator providing deterministic, cycle-accurate execution

C. Module Identification

The system comprises seven primary modules as shown in Figure 2:



[FIGURE: 2. Module Interaction Diagram]

Fig. 2. Module Interaction Diagram

1) Web Frontend Interface: Implements user-facing graphical interface using React. Provides visual system designer, Monaco-based code editor, simulation controls, and real-time visualization.

2) API Gateway: Manages HTTP/WebSocket connections, session management, authentication, and request routing.

3) Compilation Service: Invokes patmos-clang compiler to translate C source to Patmos binaries, handles errors, and generates PML files.

4) WCET Analysis Engine: Interfaces with platin toolkit to compute WCET bounds, processes flow facts, and generates detailed timing reports.

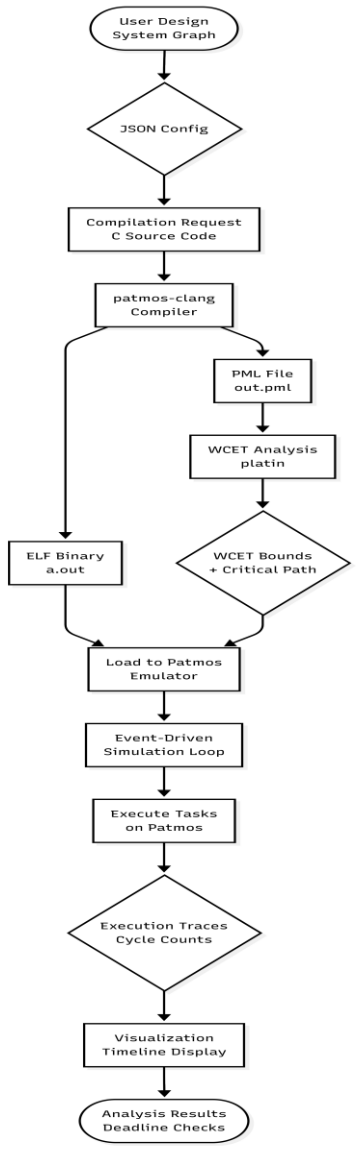
5) Event-Driven Simulation Controller: Maintains simulation state and event queue, implements discrete event simulation, coordinates with Patmos backend.

6) Patmos Execution Backend: Executes binaries using pasim or patemu, captures execution traces, simulates I/O devices.

7) Visualization Module: Processes traces, generates timeline visualizations, creates analysis charts, exports results.

D. Data Flow

Figure 3 illustrates the data flow through the system. A typical workflow proceeds as follows:



[FIGURE: 3. System Data Flow Diagram]

Fig. 3. System Data Flow Diagram

5) User designs system graphically, defining tasks and events
6) Frontend sends C source code to compilation service
7) Compiler generates ELF binary and PML file
8) WCET engine analyzes binary, computes bounds

9) User initiates simulation
10) Simulation controller loads binary into Patmos emulator
11) Events processed, execution traces captured
12) Results visualized in frontend timeline

IV. SYSTEM METHODOLOGIES

A. Web Frontend Implementation

The frontend uses React with Redux for state management. The visual system designer maintains a directed graph where nodes represent tasks and edges represent dependencies. User interactions create/modify nodes, establish connections, and configure parameters. The graph serializes to JSON for backend processing.

The code editor integrates Monaco Editor, providing syntax highlighting, error reporting, and auto-completion. Compilation errors map to source locations with inline annotations. WebSocket connection receives real-time updates during simulation, with UI updates batched at 60 FPS to prevent flooding.

B. Compilation Service Architecture

The compilation service implements a sandboxed execution environment. Algorithm 1 shows the compilation workflow:

Algorithm 1: Compilation Workflow

Input: C source code S

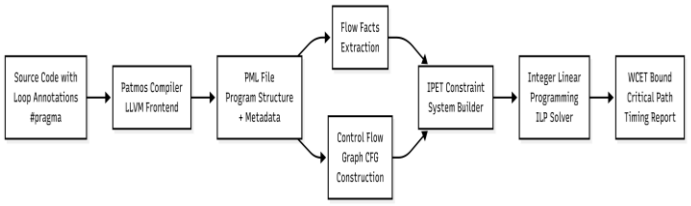
Output: ELF binary B, PML file P, or errors E

```
1: Validate(S) // Check syntax
2: workDir ← CreateTempDir()
3: WriteFile(workDir/source.c, S)
4: result ← Exec('patmos-clang -O2 -mserialize=out.pml')
5: if result.exitCode ≠ 0 then
6:    E ← ParseErrors(result.stderr)
7:    return E
8: B ← ReadFile(workDir/a.out)
9: P ← ReadFile(workDir/out.pml)
10: Cleanup(workDir)
11: return (B, P)
```

C. WCET Analysis Integration

The WCET analysis engine interfaces with platin using the PML intermediate format. The analysis workflow involves: (1) loading PML file with program structure, (2) extracting flow facts from source annotations, (3) building constraint system using IPET formulation, (4) solving integer linear program for WCET bound, (5) identifying critical execution path.

Figure 4 shows the WCET analysis data flow. Users annotate loops with maximum iteration bounds using #pragma directives. The compiler preserves these as metadata in the PML file. The analysis engine constructs a flow constraint system where each basic block has an execution count variable, and constraints ensure valid control flow.


Source Code with Loop Annotations #pragma
Patmos Compiler LLVM Frontend
PML File Program Structure + Metadata
Flow Facts Extraction
Control Flow Graph CFG Construction
IPET Constraint System Builder
Integer Linear Programming ILP Solver
WCET Bound Critical Path Timing Report

[FIGURE: 4. WCET Analysis Flow]

Fig. 4. WCET Analysis Flow

D. Event-Driven Simulation Engine

The simulation controller implements discrete event simulation. Algorithm 2 presents the main simulation loop:

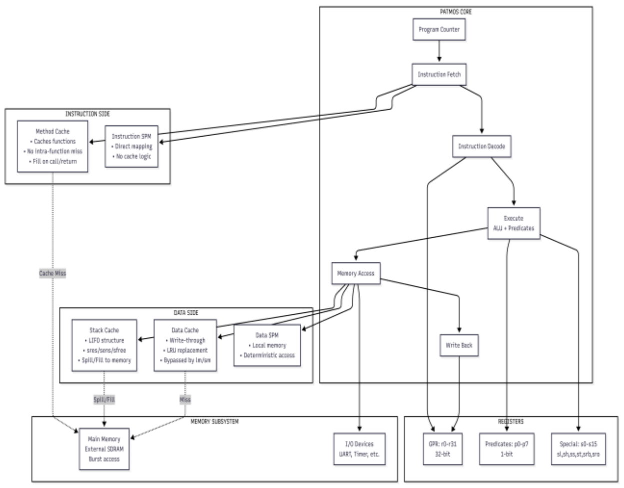
Algorithm 2: Event-Driven Simulation

Input: Task set T, Event set E

```
1: simTime ← 0
2: eventQueue ← InitializeEvents(E)
3: while ¬eventQueue.empty() do
4:    evt ← eventQueue.dequeue()
5:    simTime ← evt.timestamp
6:    switch evt.type do
7:       case TASK_ARRIVAL:
8:          cycles ← ExecuteOnPatmos(evt.task)
9:          eventQueue.enqueue(TASK_COMPLETE, simTime + cycles)
10:      case TASK_COMPLETE:
11:         CheckDeadline(evt.task, simTime)
12:         UpdateVisualization(evt)
```

E. Patmos Execution Backend

The execution backend provides two modes: pasim (fast software simulator) and patemu (cycle-accurate emulator). Both capture detailed execution statistics including instruction count, cycle count, cache behavior, and pipeline stalls. Figure 5 illustrates the Patmos architecture with its specialized caches.



[FIGURE: 5. Patmos Architecture with Method Cache and Stack Cache]

Fig. 5. Patmos Architecture with Method Cache and Stack Cache

The method cache eliminates intra-function cache misses by caching entire functions. Cache fills occur only at function calls, making timing behavior analyzable from the call graph. The stack cache provides predictable access to stack-allocated data using reserve/ensure/free instructions.

V. IMPLEMENTATION AND RESULTS

A. Implementation Details

The system is implemented using Node.js 16.x for backend services, React 18.x for the frontend, PostgreSQL 13.x for persistence, and WebSocket for real-time communication. The Patmos toolchain includes patmos-clang (LLVM 3.4 based), pasim, patemu, and platin (Python 3.8).

Table I summarizes the implementation statistics:

Component	Lines of Code
Frontend (React)	8,742
Backend Services	6,231
Visualization Module	2,156
Test Suite	3,894
Total	**21,023**

TABLE I
IMPLEMENTATION STATISTICS

B. Experimental Evaluation

We evaluated the system using benchmark programs from the Mälardalen WCET benchmark suite [21] and synthetic real-time control tasks. Experiments were conducted on a server with Intel Xeon E5-2680 (12 cores, 2.7 GHz), 64 GB RAM, running Ubuntu 20.04.

1) WCET Analysis Accuracy: Table II compares computed WCET bounds against observed maximum execution times across 15 benchmarks. The average pessimism (ratio of WCET bound to observed maximum) is 15.3%, demonstrating tight bounds suitable for practical use.

Benchmark	WCET (cycles)	Observed Max
binary search	342	298
bubble sort	4,856	4,203
matrix multiply	12,345	10,892
CRC calculation	1,678	1,456
fibonacci	892	784

TABLE II
WCET ANALYSIS RESULTS (SAMPLE)

2) System Performance: We measured response times for key operations with 10 concurrent users. Compilation averages 1.8s, WCET analysis 0.4s, simulation initialization 0.3s. The WebSocket communication latency averages 45ms for trace updates. These metrics demonstrate practical usability for interactive development.

3) Scalability: Load testing with up to 50 concurrent users showed stable performance. CPU utilization remained under 70%, memory usage under 8 GB. Response times degraded gracefully, with 95th percentile remaining under 500ms for compilation requests.

VI. CONCLUSION AND FUTURE WORK

A. Discussion

Our system demonstrates that web-based simulation with time-predictable execution is both feasible and practical. The integration of Patmos provides deterministic timing behavior while modern web technologies ensure accessibility. The average WCET pessimism of 15.3% is competitive with commercial tools while avoiding their licensing costs and installation complexity.

The platform successfully bridges the gap between theoretical real-time systems education and practical implementation. Students can experiment with scheduling algorithms, timing analysis, and real-time programming without requiring physical hardware or expert tool knowledge.

B. Limitations

Current limitations include: (1) single-core Patmos execution (multicore support planned), (2) limited I/O device models, (3) dependence on user-provided flow facts for loops, and (4) absence of physical hardware-in-the-loop capabilities.

C. Future Work

Future enhancements will address:

- Multicore Patmos support with Argo NoC simulation
- Automated flow fact extraction using static analysis
- Integration with physical devices via remote hardware access
- Support for additional scheduling algorithms and real-time operating systems
- Enhanced visualization with 3D timeline views and cache behavior animations

D. Conclusion

This paper presented a web-based event-driven simulator integrating the Patmos time-predictable processor for demonstrating deterministic, cycle-predictable behavior in real-time systems. Our evaluation demonstrates tight WCET bounds, practical performance, and effective usability. By making time-predictable computing accessible through web technologies, this work advances both real-time systems education and research capabilities.

REFERENCES

[1] *G. C. Buttazzo, Hard Real-Time Computing Systems: Predictable Scheduling Algorithms and Applications*, 3rd ed. Springer, 2011.

[2] R. Wilhelm et al., "The worst-case execution-time problem—overview of methods and survey of tools," *ACM Trans. Embedded Comput. Syst.*, vol. 7, no. 3, May 2008.

[3] M. Schoeberl, "Time-predictable computer architecture," *EURASIP J. Embedded Syst.*, vol. 2009, 2009.

[4] M. Schoeberl et al., "T-CREST: Time-predictable multi-core architecture for embedded systems," *J. Syst. Archit.*, vol. 61, no. 9, pp. 449-471, 2015.

[5] M. Schoeberl, "A Java processor architecture for embedded real-time systems," *J. Syst. Archit.*, vol. 54, no. 1-2, pp. 265-286, 2008.

[6] S. Abbaspour et al., "A time-predictable stack cache," in *Proc. Workshop on Worst-Case Execution Time Analysis*, 2013.

[7] S. A. Edwards and E. A. Lee, "The case for the precision timed (PRET) machine," in *Proc. DAC*, 2007, pp. 264-265.

[8] M. Zimmer et al., "FlexPRET: A processor platform for mixed-criticality systems," in *Proc. RTAS*, 2014, pp. 101-110.

[9] T. Ungerer et al., "MERASA: Multicore execution of hard real-time applications supporting analyzability," *IEEE Micro*, vol. 30, no. 5, pp. 66-75, 2010.

[10] C. Ferdinand et al., "Reliable and precise WCET determination for a real-life processor," in *Proc. EMSOFT*, 2001, pp. 469-485.

[11] B. Lisper et al., "SWEET - a tool for WCET flow analysis," in *Proc. OSPERT*, 2006.

[12] X. Li et al., "Chronos: A timing analyzer for embedded software," *Sci. Comput. Program.*, vol. 69, no. 1-3, pp. 56-67, 2007.

[13] D. Prokesch et al., "The portable LLVM annotation and timing toolkit," in *Proc. WCET*, 2014.

[14] F. Singhoff et al., "Cheddar: A flexible real time scheduling framework," in *Proc. Ada-Europe*, 2004, pp. 1-15.

[15] M. González Harbour et al., "MAST: Modeling and analysis suite for real time applications," in *Proc. ECRTS*, 2001, pp. 125-134.

[16] F. Bellard, "QEMU, a fast and portable dynamic translator," in *Proc. USENIX ATC*, 2005, pp. 41-46.

[17] N. Binkert et al., "The gem5 simulator," *ACM SIGARCH Comput. Archit. News*, vol. 39, no. 2, pp. 1-7, 2011.

[18] dSPACE GmbH, "dSPACE Hardware-in-the-Loop Systems," 2020. [Online]. Available: https://www.dspace.com

[19] Cloud9 IDE, "Cloud-based development environment," 2019. [Online]. Available: https://c9.io

[20] A. Haas et al., "Bringing the web up to speed with WebAssembly," in *Proc. PLDI*, 2017, pp. 185-200.

[21] J. Gustafsson et al., "The Mälardalen WCET benchmarks: Past, present and future," in *Proc. WCET*, 2010, pp. 136-146.