

CHAPTER 1

INTRODUCTION

1.1 Overview

Real-time systems demand predictable execution behavior to ensure that worst-case execution time (WCET) can be statically determined with high confidence. Traditional processors, optimized for average-case performance through complex features like speculative execution, dynamic branch prediction, and multilevel caching, introduce significant timing variability that complicates WCET analysis. This unpredictability poses fundamental challenges for safety-critical applications in automotive, aerospace, industrial automation, and medical devices where missing a deadline can have catastrophic consequences.

This project addresses these challenges by developing a web-based, event-driven simulator that offloads time-critical control logic to a Patmos-based execution backend. Patmos is a time-predictable processor specifically designed for real-time systems, featuring a static scheduled dual-issue RISC architecture that enables tight WCET bounds through deterministic, cycle-predictable behavior.

The simulator provides an accessible platform for researchers, educators, and engineers to experiment with real-time system design, test control algorithms, and validate timing requirements without requiring physical hardware. By combining the flexibility of web-based simulation with the determinism of Patmos, this project bridges the gap between theoretical real-time systems concepts and practical implementation concerns.

1.2 Description

The project consists of three primary components working in concert:

- **Web-Based Frontend:** A modern, responsive user interface built using contemporary web technologies (React/Vue.js) that allows users to design event-driven systems, configure simulation parameters, visualize execution traces, and analyze timing behavior in real-time.
- **Event-Driven Simulation Engine:** A sophisticated middleware layer that manages event queues, handles inter-component communication, orchestrates task scheduling, and maintains simulation state while interfacing between the frontend and the Patmos backend.

- **Patmos Execution Backend:** The time-predictable processor implementation that executes compiled control logic with guaranteed timing bounds, providing cycle-accurate execution traces and performance metrics that can be fed back to the simulation engine and displayed in the frontend.

The system architecture follows a layered approach where timing-critical computations are delegated to Patmos while the web interface handles user interaction, visualization, and non-time-critical processing. This separation of concerns enables the simulator to maintain both usability and timing guarantees.

1.3 Objective

The primary objectives of this project are:

- To demonstrate the feasibility and advantages of using time-predictable architectures for real-time system simulation and validation.
- To provide an accessible educational tool that helps students and practitioners understand real-time systems concepts, WCET analysis, and timing predictability without requiring specialized hardware.
- To create a research platform for experimenting with event-driven control algorithms, scheduling policies, and timing analysis techniques in a controlled, reproducible environment.
- To validate that Patmos can deliver cycle-accurate, deterministic execution suitable for hard real-time constraints while integrated into a modern web-based simulation environment.
- To enable rapid prototyping and testing of embedded control systems before committing to physical hardware implementation, reducing development time and costs.

1.4 About the Project

This project emerged from the recognition that existing simulation tools for real-time systems often sacrifice timing accuracy for flexibility or require expensive commercial licenses and specialized knowledge. The open-source Patmos processor, developed as part of the T-CREST project (Time-predictable Multi-Core Architecture for Embedded Systems), provides a foundation for creating timing-predictable embedded systems.

Patmos distinguishes itself through several key architectural features:

- Split cache architecture with separate instruction and data caches, including a method cache for functions and a stack cache for stack frames, eliminating cache interference between different memory access patterns.
- Static dual-issue pipeline that exposes instruction-level parallelism through compiler scheduling rather than dynamic hardware mechanisms, making timing behavior analyzable at compile time.
- Predicated execution support that eliminates branch misprediction penalties, a major source of timing variability in conventional processors.
- LLVM-based compiler toolchain with integrated WCET analysis tools (platin and aiT support) that enable developers to obtain timing guarantees at compilation time.

The project leverages these capabilities to create a simulation environment where users can design event-driven control systems with graphical tools, write control logic in C that compiles to Patmos binaries, simulate system behavior with realistic timing, and analyze WCET bounds and schedulability automatically.

1.5 Organization of the Project Report

This project report is organized into four comprehensive chapters:

- Chapter 1 (Introduction) provides the context, objectives, and overview of the project, establishing the motivation for developing a web-based simulator with Patmos backend and explaining the fundamental concepts of time-predictable computing.
- Chapter 2 (Literature Survey) reviews existing research and technologies in real-time systems, time-predictable architectures, simulation methodologies, and WCET analysis techniques, positioning this project within the broader academic and industrial context.
- Chapter 3 (System Requirements and Design) details the problem definition, existing system limitations, proposed system architecture, module identification, hardware and software requirements, and comprehensive design process with architectural diagrams and component interactions.
- Chapter 4 (System Methodologies) provides detailed explanations of each module's implementation, including algorithms, data structures, communication protocols, and integration strategies that bring the system components together into a cohesive simulator.

CHAPTER 2

LITERATURE SURVEY

This chapter examines the theoretical foundations and related work that inform the design and implementation of our web-based event-driven simulator with Patmos backend. We survey key contributions across real-time systems theory, time-predictable architectures, WCET analysis methodologies, and simulation frameworks.

2.1 Title : Automatic Generation of Simulators for Processors Enhanced for Security in Virtualization

Author : Swapneel C. Mhatre and Priya Chandran

Year : 2025

All new computer architectures need to be performance evaluated for acceptance and simulation is the most widely used method for evaluation of new processor designs. Sharing resources with virtualization raised many security concerns, leading to the development of processors that are enhanced for security in virtualization. The simulators for processors enhanced for security in virtualization need to perform simulation of hypervisor instructions, simulation of security in virtualization, and simulation of new instructions. However, a simulator with all of these three features is not found in the literature. Hence, this paper proposes an approach for the simulation of processors enhanced for security in virtualization that provides all these three features. For user convenience, the simulators are generated automatically from the target processor specifications using a simulator generator. The paper also proposes an approach for simulating a new pipeline with a designer-specified number of stages with automatic detection of pipeline hazards and automatic stalling or flushing of the pipeline on detection of hazards. To demonstrate the use of the simulator generator and the generated simulator, three case studies are considered - simulation of RISC-V with

HyperWall, simulation of RISC-V with bit-serial dot-product unit, and simulation of RISC-V with Galois Field arithmetic extension. The paper concludes that the proposed approaches help in accurately simulating the overhead due to security in virtualization and also in providing flexibility to the designer to simulate the desired processor configurations.

2.2 Title : Towards Lingua Franca on the Patmos Processor**Author** : Ehsan Khodadad, Luca Pezzarossa, and Martin Schoeberl**Year** : 2024

Real-time embedded systems demand higher reliability than any other computer systems. These systems require special modeling paradigms to satisfy time constraints. This paper introduced a design method by combining T-CREST, a time-predictable multi-core hardware, with Lingua Franca, a coordination framework that generates deterministic time-predictable code. T-CREST is a complete time-predictable multicore embedded system consisting of a set of processors connected by two Network-onChips. The processor used inside the T-CREST embedded system is Patmos, a RISC time-predictable processor specifically designed for real-time embedded systems. Lingua Franca is a coordination language that can generate C, C++, Python, TypeScript, and Rust and integrate with legacy codes. The main contributions of this paper are the integration of Lingua Franca into a single-core Patmos processor and performing preliminary experiments to prove the correct functionality. The authors executed a Lingua Franca piece of software on T-CREST platform and performed preliminary experiments demonstrating its correct functionality. The results show that combining a time-predictable hardware platform with a coordination framework that generates deterministic code creates a complete software and hardware design framework dedicated to safety-critical systems.

2.3 Title : Simulators for Processors Used in Virtualization: A Survey**Author** : Swapneel C. Mhatre, Priya Chandran**Year** : 2025

This paper presents a detailed survey of architectural simulators used for processors in virtualized environments. With the growing importance of cloud computing and virtualization, accurate processor simulation has become essential for evaluating performance, security, and architectural enhancements. The authors extend existing simulator classifications by considering the simulation of hypervisor instructions, which are often ignored in earlier studies. The paper identifies major challenges in simulating processors enhanced for virtualization security, such as timing accuracy, hypervisor overhead, and memory virtualization. It also analyzes resolved and unresolved issues in current simulators. The study concludes that although significant progress has been made, many challenges remain for achieving precise timing simulation in virtualizationenabled processors.

2.4 Title : Timing-Aware Analysis of Shared Cache Interference for Non-Preemptive

Scheduling

Author : Thilo L. Fischer, Heiko Falk**Year** : 2024

This paper proposes a timing-aware analysis approach to evaluate shared cache interference in multi-core real-time systems under non-preemptive scheduling. Since shared last-level caches can cause unpredictable inter-core interference, accurately estimating worst-case execution time (WCET) becomes challenging. The authors model cache accesses using event-arrival curves to quantify interference over time and introduce a data-flow analysis to classify cache accesses as hits or potential misses. The proposed method provides tighter WCET bounds compared to traditional cache analysis techniques. Experimental results on multi-core systems demonstrate significant improvements in WCET estimation accuracy, making the approach effective for realtime systems requiring high predictability.

2.5 Title : Patmos: A Time-Predictable Microprocessor

Author : Martin Schoeberl, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, Daniel

Prokesch

Year : 2018

This paper introduces Patmos, a time-predictable RISC microprocessor specifically designed for real-time and safety-critical embedded systems where tight and analyzable worst-case execution time (WCET) bounds are essential. Unlike conventional processors optimized for average-case performance, Patmos is architected with WCET predictability as the primary design goal. The processor employs a statically scheduled dual-issue pipeline in which all instruction latencies are explicitly visible at the instruction set architecture (ISA) level, eliminating hidden timing behaviors caused by speculation, dynamic scheduling, or variable-latency instructions. To simplify WCET analysis, Patmos integrates specially designed memory components such as a method cache for instructions, a stack cache for stack-allocated data, and a separate data cache, enabling predictable memory access patterns. The paper also presents a WCET-aware compiler based on LLVM that cooperates closely with the hardware to optimize programs for worst-case performance rather than average-case speed. Additionally, the authors discuss the integration of Patmos into the T-CREST multicore platform, where time-division multiplexing ensures predictable shared-memory access. Experimental evaluation demonstrates that Patmos achieves competitive performance while significantly reducing WCET overestimation, proving its suitability for real-time systems requiring strong timing guarantees.

2.6 Title : ForSyDe on the Patmos Processor

Author : Ehsan Khodadad, Ingo Sander, Luca Pezzarossa, Martin Schoeberl

Year : 2025

This paper presents an integrated hardware–software design approach that combines the ForSyDe (Formal System Design) modeling framework with the Patmos timepredictable processor to support the development of safety-critical real-time embedded systems. ForSyDe provides a formal, model-of-computation-based methodology that allows designers to describe

system behavior at a high level of abstraction using synchronous and data-flow models, ensuring correctness by construction. The authors demonstrate how ForSyDe models can be systematically translated and executed on the T-CREST multicore platform, where Patmos serves as the processing core. The work focuses on maintaining predictability throughout the design flow, from high-level modeling to executable C code running on time-predictable hardware. Several case studies and preliminary experiments are presented, including digital signal processing applications, to validate the feasibility of the approach. Worst-case execution time (WCET) analysis is performed using the Platin WCET analysis tool, showing that the generated code preserves analyzability and predictable timing behavior. The results highlight that combining formal modeling with time-predictable processor architectures significantly reduces design complexity while ensuring timing correctness, making the approach well-suited for safety-critical domains such as avionics, automotive, and industrial control systems.

2.7 Title : Designing Predictable Cache Coherence Protocols for Multi-Core Real-Time Systems

Author : Anirudh Mohan Kaushik, Mohamed Hassan, Hiren Patel

Year : 2021

This paper addresses the challenge of achieving timing predictability in cache coherence protocols for multi-core real-time systems. Conventional cache coherence mechanisms are optimized for average-case performance, which introduces unpredictable memory access delays and complicates worst-case execution time (WCET) analysis. The authors propose predictable cache coherence protocols designed to provide bounded and analyzable memory access latencies. The approach focuses on simplifying coherence state transitions and controlling interference between cores to improve timing predictability. Experimental evaluation shows that the proposed protocols significantly reduce timing variability while maintaining acceptable performance, making them suitable for safety-critical real-time applications.

2.8 Title : Design of a Time-predictable Multicore Processor: The T-CREST Project

Author : Martin Schoeberl, Florian Brandner, Wolfgang Puffitsch, Stefan Hepp, Benedikt Huber, Daniel Prokesch

This paper presents the T-CREST project, which focuses on designing a fully time-predictable multicore processor platform for real-time and safety-critical embedded systems. Unlike conventional multicore processors optimized for average-case performance, T-CREST is designed to ensure analyzable and tight worst-case execution time (WCET) guarantees. The architecture consists of multiple Patmos time-predictable processor cores interconnected using two predictable Network-on-Chip (NoC) structures for data and instruction communication.

Time-division multiplexing is employed to control shared resource access and eliminate unpredictable interference between cores. The paper demonstrates that the proposed architecture significantly improves timing predictability while maintaining acceptable performance,

making T-CREST suitable for domains such as avionics, automotive, and industrial control systems.

2.9 Title : FPGA Hardware–Software Co-Design for Real-Time Embedded Systems

Author : M. A. Erdoğan and F. N. Demir

Year : 2025

This paper discusses an FPGA-based hardware–software co-design approach aimed at meeting the strict performance requirements of real-time embedded systems. It explains how combining the flexibility of software with the parallel processing power of FPGA hardware can significantly improve execution speed, reduce latency, and lower power consumption. The authors describe co-design methodologies, hardware–software partitioning strategies, and the use of HDLs and high-level synthesis tools. A real-time robot localization case study is presented to demonstrate practical feasibility, showing improved performance compared to software-only solutions while maintaining efficiency and adaptability. The work concludes that FPGA-based co-design is highly suitable for applications such as image processing, robotics, and other safety- and time-critical embedded systems.

2.10 Title : Time-predictable Computer Architecture

Author : Schoeberl

Year : 2009

Schoeberl's foundational work argues 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 paper introduces the concept of

time-predictability as a first-class architectural concern, demonstrating that timing analyzability and performance are not necessarily conflicting goals. This research establishes the philosophical and technical foundation for Patmos, showing that architectural features specifically designed for WCET analysis can achieve competitive performance while dramatically simplifying timing verification.

Key contributions include the identification of architectural features that introduce timing anomalies, the proposal of alternative designs that maintain predictability, and empirical evidence that time-predictable architectures can be practical for real-world applications. This work directly informs our choice of Patmos as the execution backend.

CHAPTER 3

SYSTEM REQUIREMENTS AND DESIGN

3.1 Problem Definition

Real-time embedded systems require rigorous timing validation to ensure that all deadlines are met under worst-case conditions. However, several fundamental problems hinder effective development and verification:

- **Timing Unpredictability:** Modern general-purpose processors employ sophisticated performance optimizations (branch prediction, out-of-order execution, multilevel caches) that create substantial timing variability, making WCET analysis extremely conservative or even intractable.
- **High Entry Barriers:** Developing and testing real-time systems typically requires expensive hardware platforms, specialized debugging equipment, and extensive domain expertise, limiting accessibility for education and early-stage research.
- **Inadequate Simulation Tools:** Existing simulation frameworks either sacrifice timing accuracy for ease of use or provide cycle-accurate simulation only for specific proprietary platforms without supporting systematic WCET analysis.
- **Limited Integration:** Tools for real-time system development (compilers, simulators, WCET analyzers, schedulability tests) typically exist as separate utilities with incompatible data formats, requiring manual integration and error-prone data transformation.

These problems result in extended development cycles, late discovery of timing violations, over-provisioning of computational resources due to pessimistic analysis, and difficulty in teaching real-time systems concepts effectively.

3.2 Existing System

Current approaches to real-time system simulation and validation fall into several categories, each with significant limitations:

3.2.1 High-Level Scheduling Simulators

Tools like Cheddar, MAST, and SchedCAT provide abstract task models where developers specify periods, deadlines, and estimated execution times. These tools perform schedulability analysis and can simulate task scheduling under various policies (RM, EDF, etc.). However, they do not execute actual code, requiring developers to estimate execution times separately. The disconnect between high-level task models and low-level implementation details often leads to inaccurate timing predictions and integration problems.

3.2.2 Instruction Set Simulators

Processor-specific simulators (ARM Cycle Model, QEMU, Gem5) can execute compiled binaries and provide cycle-accurate timing for specific architectures. While these tools offer detailed execution traces, they simulate processors designed for average-case performance, making WCET analysis extremely complex. Most simulators are command-line tools without graphical interfaces or integrated development environments, limiting their accessibility.

3.2.3 Hardware-in-the-Loop Platforms

Platforms like dSPACE and NI VeriStand connect real embedded controllers to simulated plant models, providing realistic execution environment for control algorithms. These systems offer high fidelity but require expensive hardware, are difficult to share remotely, and typically don't provide systematic WCET analysis or formal timing verification capabilities.

3.2.4 Limitations Summary

Existing systems suffer from fragmentation (tools don't interoperate), accessibility issues (requiring specialized hardware or commercial licenses), timing inaccuracy (abstract models) or unpredictability (conventional processors), and lack of integration (separate compilation, simulation, and analysis steps).

3.3 Proposed System

Our proposed system addresses the limitations of existing approaches by providing an integrated, web-based simulation platform that combines:

- **Accessible Web Interface:** Users access the simulator through standard web browsers without requiring local installation of specialized tools or toolchains.
- **Graphical System Design:** Visual editors allow users to specify event-driven system architectures, define tasks and their dependencies, configure sensors and actuators, and establish communication patterns without writing boilerplate code.
- **Integrated Development Environment:** Browser-based code editor with syntax highlighting, compilation error feedback, and direct integration with the Patmos toolchain (patmos-clang compiler and platin analysis tool).
- **Time-Predictable Execution:** All time-critical control logic executes on Patmos emulator or hardware, providing deterministic, cycle-accurate timing that can be formally analyzed.
- **Automatic WCET Analysis:** The system automatically computes WCET bounds using platin and presents results graphically, highlighting potential timing violations before deployment.

- **Real-Time Visualization:** Interactive timeline view showing task activations, execution traces, cache behavior, and inter-task communication, enabling users to understand system timing behavior intuitively.

3.3.1 Objectives of Proposed System

- **Educational Accessibility:** Lower barriers to learning real-time systems concepts by providing immediate, interactive feedback without hardware requirements.
- **Research Enablement:** Facilitate experimentation with scheduling algorithms, timing analysis techniques, and real-time programming patterns in a reproducible environment.
- **Industrial Prototyping:** Enable rapid development and validation of control algorithms before committing to hardware implementation and deployment.
- **Timing Verification:** Provide automatic detection of deadline violations, priority inversions, and resource conflicts through integrated schedulability analysis.

3.4 Module Identification

The system architecture is decomposed into seven primary modules:

Module 1: Web Frontend Interface

Implements the user-facing graphical interface using React framework. Provides visual system designer with drag-and-drop components, code editor with syntax highlighting and error reporting, simulation control panel, real-time execution timeline, WCET analysis results display, and configuration management.

Module 2: API Gateway and Session Management

Handles HTTP/WebSocket connections between frontend and backend services. Manages user sessions, authentication, and authorization. Routes API requests to appropriate backend services. Implements rate limiting and request validation.

Module 3: Compilation Service

Invokes patmos-clang compiler to translate C source code to Patmos ELF binaries. Handles compiler errors and warnings, returning formatted messages to frontend. Generates PML (Patmos Modeling Language) files for WCET analysis. Manages compiled binary storage and versioning.

Module 4: WCET Analysis Engine

Interfaces with platin toolkit to compute WCET bounds. Processes PML files and binary executables. Generates flow facts from code annotations. Produces detailed timing reports with execution path information. Identifies critical paths and timing bottlenecks.

Module 5: Event-Driven Simulation Controller

Maintains simulation state and event queue. Implements discrete event simulation algorithm. Handles event scheduling, priorities, and ordering. Manages simulated time advancement. Coordinates communication between simulated components and Patmos backend.

Module 6: Patmos Execution Backend

Executes compiled Patmos binaries using either pasim (software simulator) or patemu (hardware emulator). Captures cycle-accurate execution traces. Provides I/O device simulation for interrupts and peripheral interactions. Returns execution metrics (cycles consumed, cache statistics, stack usage) to simulation controller.

Module 7: Visualization and Analysis Module

Processes execution traces and timing data for display. Generates interactive timeline visualizations. Produces cache behavior diagrams. Creates schedulability analysis charts. Exports simulation results in various formats (CSV, JSON, PDF reports).

3.5 System Requirements

3.5.1 Hardware Requirements

Server-Side Hardware:

- Processor: Multi-core x86-64 processor (4+ cores recommended)
- RAM: Minimum 8 GB, 16 GB recommended for concurrent users
- Storage: 50 GB SSD for OS, tools, and user data
- Network: Gigabit Ethernet connection

Client-Side Hardware:

- Any device with modern web browser (desktop, laptop, or tablet)
- Minimum 4 GB RAM
- Display resolution: 1920x1080 or higher recommended

3.5.2 Software Requirements

Server-Side Software:

- Operating System: Linux (Ubuntu 20.04 LTS or later recommended)
- Patmos Toolchain: patmos-clang, pasim, patemu, platin
- Node.js: Version 16.x or later
- Python: Version 3.8 or later (for platin)
- Database: PostgreSQL 13.x or later

- Web Server: Nginx 1.18 or later

Client-Side Software:

- Modern web browser: Chrome 90+, Firefox 88+, Safari 14+, or Edge 90+
- JavaScript enabled
- WebSocket support

3.6 Design Process and Explanation

3.6.1 Overall System Architecture

The system follows a three-tier architecture pattern:

- Presentation Tier: Web frontend built with React, handling user interaction and visualization
- Application Tier: Node.js backend services implementing business logic, simulation control, and toolchain integration
- Execution Tier: Patmos simulator/emulator providing deterministic execution environment

3.6.2 Communication Flow

The system employs hybrid communication strategies:

- RESTful API: For stateless operations like compilation requests, configuration retrieval, and result queries
- WebSocket: For real-time simulation updates, streaming execution traces, and bidirectional event communication
- Message Queue: Internal service communication using message-passing patterns for asynchronous task processing

3.6.3 Data Flow Architecture

The typical workflow proceeds as follows:

1. User designs system graphically in web frontend, defining tasks, events, and control logic
2. Frontend sends compilation request with C source code to API gateway
3. Compilation service invokes patmos-clang, generating ELF binary and PML file
4. WCET analysis engine processes PML file using platin, computing timing bounds
5. Results returned to frontend for review; user initiates simulation
6. Simulation controller loads compiled binary into Patmos emulator

7. Event-driven simulation executes, dispatching events to Patmos backend
8. Execution traces and metrics streamed back via WebSocket
9. Visualization module processes traces and updates timeline display
10. Post-simulation analysis generates reports and identifies timing violations

3.6.4 Security Considerations

Security mechanisms include:

- Sandboxed compilation: Compiler executes in isolated container with resource limits
- Input validation: All user inputs sanitized to prevent injection attacks
- Resource quotas: Per-user limits on compilation time, simulation duration, and storage
- Authentication: JWT-based token authentication for API access

CHAPTER 4

SYSTEM METHODOLOGIES

This chapter provides detailed explanations of the implementation methodologies for each system module, including algorithms, data structures, and processing logic.

4.1 Module 1: Web Frontend Interface

4.1.1 Component Architecture

The frontend follows React component-based architecture with Redux for state management. Key components include `SystemDesigner` (drag-and-drop canvas), `CodeEditor` (Monaco-based editor), `SimulationController` (play/pause/step controls), `TimelineViewer` (execution visualization), and `AnalysisPanel` (WCET results display).

4.1.2 Visual System Design Algorithm

The visual designer maintains a graph data structure where nodes represent tasks/components and edges represent dependencies or communication channels. Drag-and-drop operations create nodes, connections establish edges, and property panels configure node parameters. The graph is serialized to JSON for backend processing and persisted in local storage for recovery.

4.1.3 Real-Time Update Mechanism

WebSocket connection receives simulation events and execution traces. Updates are batched (60 FPS) to prevent UI flooding. Timeline viewport implements virtual scrolling for efficient rendering of large trace datasets. State updates use immutable data structures to enable efficient React re-rendering.

4.2 Module 2: Compilation Service

4.2.1 Compilation Workflow

Upon receiving compilation request: (1) Source code validated for syntax, (2) Temporary working directory created, (3) `patmos-clang` invoked with appropriate flags (`-O2` optimization, `-mserialize` for PML generation), (4) Compiler output captured and parsed, (5) On success, ELF binary and PML file returned; on failure, error messages formatted and returned, (6) Temporary files cleaned up.

4.2.2 Error Handling and Reporting

Compiler errors parsed to extract filename, line number, column, and message. Errors mapped back to source code locations in editor. Warnings categorized by severity. Compilation timeout enforced (30 seconds default) to prevent resource exhaustion.

4.3 Module 3: WCET Analysis Engine

4.3.1 Analysis Pipeline

The analysis engine: (1) Loads PML file containing program structure and annotations, (2) Invokes platin with IPET-based analysis, (3) Processes flow facts from source annotations, (4) Computes WCET bound for specified entry point, (5) Generates detailed timing report with critical path identification, (6) Extracts cache statistics and memory access patterns.

4.3.2 Flow Fact Integration

Users annotate loops with maximum iteration bounds using pragmas. The analysis engine extracts these annotations during PML processing. Flow facts combined with program structure to constrain IPET formulation. Missing flow facts detected and reported to user with suggestions.

4.4 Module 4: Event-Driven Simulation Controller

4.4.1 Discrete Event Simulation Algorithm

The simulation maintains: (1) Current simulation time (cycles), (2) Priority queue of pending events ordered by timestamp, (3) System state including task status and resource allocation. Main simulation loop: Dequeue next event, advance simulation time to event timestamp, process event by invoking appropriate handler, handlers may schedule new events, continue until event queue empty or user stops simulation.

4.4.2 Event Types and Handlers

Supported event types: TASK_ARRIVAL (periodic task activation), TASK_COMPLETE (task finishes execution), INTERRUPT (external interrupt occurs), TIMER_EXPIRE (timer deadline reached), MESSAGE_SEND/RECEIVE (inter-task communication). Each event type has dedicated handler that updates system state and may interact with Patmos backend for execution.

4.4.3 Integration with Patmos Backend

When task execution event processed: (1) Task parameters passed to Patmos emulator, (2) Emulator executes binary for specified entry point, (3) Execution trace captured including cycle count and cache events, (4) Simulation time advanced by actual execution cycles, (5) Trace data returned to frontend for visualization.

4.5 Module 5: Patmos Execution Backend

4.5.1 Simulator vs Emulator Selection

Two execution backends available: pasim (software simulator, faster but less accurate) and patemu (hardware emulator, cycle-accurate but slower). Selection based on user preference and analysis requirements. Emulator mode recommended for timing validation; simulator mode suitable for functional testing.

4.5.2 Execution Trace Capture

Both pasim and patemu configured to output execution statistics: instruction count, cycle count, cache hit/miss rates, method cache fills, stack cache spills/fills. Traces parsed and structured for storage and visualization. Critical path identification highlights instructions on WCET path.

4.5.3 I/O Device Simulation

Simulated I/O devices include: Timer (configurable periodic interrupts), UART (character I/O for debugging), GPIO (digital I/O for control signals), Interrupt controller (manages multiple interrupt sources). Device models integrated with Patmos emulator, responding to I/O instructions with deterministic timing.

4.6 Module 6: Visualization and Analysis Module

4.6.1 Timeline Visualization Algorithm

Timeline displays horizontal bars for each task showing: Task arrival, task execution (colored segments indicating active execution), task completion, idle periods. Vertical axis represents different tasks/cores. Horizontal axis shows time in cycles. Zoom and pan controls allow detailed examination. Tooltips provide detailed information on hover.

4.6.2 Cache Behavior Visualization

Cache state displayed as: Method cache content (functions currently cached), Stack cache state (allocated stack frames), Data cache status (accessed memory regions). Color coding indicates cache hits (green) versus misses (red). Cache events annotated on timeline showing when fills/spills occur.

4.6.3 Schedulability Analysis

Post-simulation analysis checks: All tasks met deadlines, No priority inversions occurred, Resource utilization within bounds, Response time distributions. Violations highlighted with detailed explanations. Suggestions provided for resolving timing problems (adjusting priorities, periods, or optimizing code).

4.6.4 Report Generation

Comprehensive reports include: Executive summary with key metrics, Detailed timing analysis per task, WCET bounds versus observed execution times, Cache performance statistics, Schedulability analysis results, Execution timeline visualizations. Reports exportable as PDF for documentation or presentation.

4.7 Integration and Testing Strategy

Module integration follows bottom-up approach: (1) Individual modules tested in isolation with mock interfaces, (2) Patmos backend integrated and validated against known benchmarks, (3) Compilation and WCET analysis services added, (4) Simulation controller integrated with backend, (5) Frontend connected for end-to-end testing, (6) Performance testing with concurrent users, (7) Security testing and penetration testing performed.

Test cases cover: Functional correctness (compilation, execution, analysis), Timing accuracy (comparing simulated versus actual Patmos timing), Scalability (multiple concurrent users and large systems), Robustness (error handling and recovery), Usability (user interface and workflow validation).

4.8 Summary

This chapter has presented the detailed methodologies for implementing each system module. The modular architecture enables independent development and testing while the well-defined interfaces ensure seamless integration. The combination of web-based accessibility, graphical system design, automatic compilation and WCET analysis, and cycle-accurate Patmos execution provides a comprehensive platform for real-time system development and validation. The systematic approach to visualization and analysis enables users to understand timing behavior intuitively and identify potential problems early in the development cycle.