Design and Verification of a Finite State Machine-Based Watchdog Controller for FPGA-Driven Process Supervision

Senthilmurugan S.[1], Abishek Prasad C. S.[2], Govindaraj P.[3], Guru Aakash P. S.[4]
[1,2,3,4]*Student, Dept. of Electronics and Communication Engineering, SRM Valliammai Engineering College, Kattankulathur – 603 203, India*

***Abstract*—Desktop operating systems lack a dedicated hardware mechanism to detect and stop a single application that has stopped responding while everything else stays running. This paper presents Phase 1 of a hardware–software co-designed Task Kill Controller (TKC): the design, RTL implementation, and functional simulation of the Finite State Machine (FSM) that acts as the decision core of the system. The FSM behaves as a programmable watchdog controller on a Zynq-7000 FPGA — it receives an 8-bit process utilisation score, compares it against a tunable threshold, and asserts either an ALLOW or a DENY output in a fixed number of clock cycles. Synthesis on the xc7z020clg484-1 device confirms minimal resource use, 0.003 W dynamic power, and zero timing violations at 100 MHz. Phase 2 will complete the prototype by adding UART communication and a Python-based host monitoring agent.**

***Index Terms*—FPGA, Finite State Machine, Watchdog Controller, Process Supervision, Zynq-7000, Hardware–Software Co-Design, Vivado, Embedded Systems.**

I. INTRODUCTION

A runaway application — one that has frozen internally but keeps consuming CPU time and memory — is a common failure mode on desktop computers. The usual fix, opening the Task Manager or pressing ALT+F4, only works if the graphical interface is still responsive enough to serve the user. When the system is truly saturated, that is often no longer the case. Embedded systems have addressed an equivalent problem for decades using watchdog timer circuits [1],[2]: dedicated hardware that monitors execution progress independently of the software it supervises and can trigger a corrective action within a bounded time.

Desktop platforms, however, have no comparable mechanism that can target a single offending application selectively. Research on FSM-based configurable watchdog timers has demonstrated that a threshold-comparison state machine is compact and reliable in programmable logic [3],[4]. Work on hardware–software fault tolerance [5] further supports separating the supervisory decision from the software being supervised. This paper builds on those principles to design a hardware decision module — a five-state FSM on a Zynq-7000 FPGA — that evaluates a numeric process score and produces a deterministic ALLOW or DENY verdict.

This is Phase 1 of a two-phase project. Phase 1 covers the RTL design of the FSM, its synthesis on the xc7z020clg484-1 device, and simulation results. Phase 2 will add a UART communication channel and a Python monitoring agent on the host PC to complete the end-to-end Task Kill Controller prototype.

II. MOTIVATION AND PROBLEM CONTEXT

Software-based task management has two structural weaknesses under high load. First, the task manager competes for the same CPU time as the problematic application — when neither gets scheduled promptly, the problem persists. Second, even when the task manager launches, terminating a process requires several user interactions, adding seconds of delay to an already degraded system.

Hardware watchdog timers avoid both problems [1],[6]. They operate on their own clock domain, carry no graphical state, and respond within a guaranteed number of cycles. The limitation of existing watchdog designs is granularity: they reset the whole system rather than singling out one process. The FSM described in this paper introduces selective, score-based granularity while preserving the determinism that makes hardware supervision attractive.

III. FSM DESIGN — WATCHDOG CONTROLLER

The Task Kill Controller FSM is a synchronous design clocked at 100 MHz. Its inputs are an 8-bit signal `score[7:0]` and a one-cycle `valid` strobe. Its output is a registered 1-bit signal `allow`. The machine cycles through five states as described below.

A. State Descriptions

- IDLE — The default resting state. The FSM waits here until a valid strobe arrives. No output is driven.
- RECEIVE — The score value is latched into an internal 8-bit register on the rising clock edge that follows the valid pulse.
- CHECK — The latched score is compared unsigned against the threshold. The comparison is strictly greater-than: score > threshold.
- ALLOW — Reached when score > threshold. The allow output is registered as logic 1 for one clock cycle before the machine returns to IDLE.
- DENY — Reached when score ≤ threshold. The allow output is registered as logic 0 for one clock cycle before the machine returns to IDLE.

The state path is strictly linear: IDLE → RECEIVE → CHECK → ALLOW or DENY → IDLE. Because there are no loops in the decision path, the worst-case latency from score arrival to verdict output is exactly four clock cycles — 40 ns at 100 MHz. This bounded latency mirrors the core property of hardware watchdog timers [3],[6]: the response always arrives within a known deadline.

B. Threshold Comparator

The threshold is implemented as a `localparam` with a default value of 50. The score fed to the FSM is the sum of CPU utilisation percentage and memory utilisation percentage for the target process, giving a range of 0 to 200. A threshold of 50 means the FSM will assert ALLOW only when the combined resource burden exceeds 50 per cent. The parameter can be changed at synthesis time without altering any RTL logic, making the controller tunable for different deployment conditions [3],[4].

C. RTL Implementation

The RTL is written in synthesisable Verilog. A single `always @(posedge clk)` block handles both state transitions and output registration in the same clock edge. Using a registered output rather than a combinational one prevents glitches on the allow net during state changes. Reset is synchronous and active-high, following VLSI reset design guidelines [7]: the FSM returns cleanly to IDLE on reset, ensuring a stable power-on state each time the device is programmed.

TABLE I. FSM STATE TRANSITION TABLE

Current State	Condition	Next State	allow Output
IDLE	valid = 1	RECEIVE	—
IDLE	valid = 0	IDLE	—
RECEIVE	always	CHECK	—
CHECK	score > 50	ALLOW	—
CHECK	score ≤ 50	DENY	—
ALLOW	always	IDLE	1
DENY	always	IDLE	0

IV. FPGA IMPLEMENTATION

The FSM was synthesised and implemented in Xilinx Vivado 2023.1 targeting the xc7z020clg484-1 — the Zynq-7000 SoC device on the project development board. Only the programmable logic (PL) fabric is active at this stage; the embedded ARM core is disabled and contributes nothing to the resource count.

A. Resource Utilisation

The post-synthesis netlist resolves to a small cluster of `FDRE` flip-flop primitives for the state register and the latched score, plus a few LUT cells for the comparator and next-state logic. No block RAM, DSP slices, or PLL resources are required. This footprint is consistent with results published for comparable FSM-based watchdog designs on similar device families [2],[3].

B. Timing

With a 10 ns clock constraint, the design closes timing with positive slack on all 284 registered endpoints. The longest combinational path is the 8-bit unsigned comparator in the CHECK state, which still leaves the critical path well below the clock period. No timing violations are present [4].

C. Power

Dynamic switching power is below 0.003 W. Device static leakage dominates the total on-chip power figure, which is an intrinsic characteristic of the xc7z020 technology node rather than a consequence of the design. The active logic draws power almost entirely through clock distribution, matching results reported for compact FSM designs on similar platforms [2].

TABLE II. SYNTHESIS POWER SUMMARY

Parameter	Value
Total On-Chip Power	0.109 W
Dynamic Power	0.003 W (2%)
Clocks	0.002 W (78% of dynamic)
Logic	<0.001 W
Device Static	0.106 W (98%)
Junction Temperature	26.3 °C
Thermal Margin	58.7 °C

D. Methodology Compliance

Vivado's Report Methodology returned zero violations. The clocking structure, synchronous reset topology, and I/O assignment all satisfy the tool's recommended guidelines. Clean synchronous reset design [7] was implemented throughout to guarantee a known power-on state on every board startup.

V. SIMULATION AND VERIFICATION

Functional verification used the Vivado behavioural simulator. A self-checking testbench applied all 256 possible score values (0–255) in sequence and verified the `allow` output against the expected result for each input. The FSM passed every check with no mismatches.

TABLE III. SIMULATION RESULTS — BOUNDARY CASES

Score	Threshold	FSM Output	Pass?
7	50	DENY (allow = 0)	Yes
49	50	DENY (allow = 0)	Yes
50	50	DENY (allow = 0)	Yes
51	50	ALLOW (allow = 1)	Yes
100	50	ALLOW (allow = 1)	Yes
200	50	ALLOW (allow = 1)	Yes

The key results from simulation are: *(i)* The FSM correctly implements a strictly greater-than rule — a score of 50 produces DENY while a score of 51 produces ALLOW. *(ii)* State transitions complete in exactly four clock cycles from `valid` assertion to `allow` assertion. *(iii)* No glitches or hazards are present on the output during state transitions [3].

VI. PLANNED PHASE 2 EXTENSIONS

Phase 2 will connect the FSM to a real host PC by adding three components:

- A UART Receiver module, parameterised for 115,200 baud at a 100 MHz clock, that accepts the score byte from the PC and presents it to the FSM with a valid strobe.
- A UART Transmitter module that serialises the FSM's allow output as ASCII 'A' (0x41) for ALLOW or 'D' (0x44) for DENY and sends it back to the host.
- A Python monitoring agent on the host PC that reads CPU and memory usage for all running processes using the psutil library, picks the highest-scoring process, and dispatches its score over a USB-to-serial bridge when the user presses CTRL+SHIFT+A.

The FSM designed in Phase 1 will remain unchanged. It connects directly to the UART receiver's `data[7:0]` and `valid` outputs. The interface contract — one score byte in, one verdict byte out — was chosen to keep both the hardware and software sides independently verifiable [5].

VII. CONCLUSION

This paper presented Phase 1 of the Task Kill Controller: the design, synthesis, and functional simulation of a five-state Finite State Machine that acts as a programmable watchdog controller for individual process supervision on a Zynq-7000 FPGA. Synthesis on the xc7z020clg484-1 device confirmed zero timing violations at 100 MHz, 0.003 W dynamic power, minimal logic usage, and a clean Vivado Methodology Report. Simulation across all 256 input values showed correct ALLOW–DENY behaviour with a worst-case decision latency of four clock cycles.

The design is self-contained and ready to receive the UART interface and host agent planned for Phase 2. Together, the two phases will form a complete, hardware-enforced process termination system that operates deterministically regardless of host OS load — a capability that concepts from hardware

watchdog timer research [1],[2],[6] show is both practical and achievable on commodity FPGA hardware.

ACKNOWLEDGMENT

The authors thank the faculty of the Department of Electronics and Communication Engineering, SRM Valliammai Engineering College, for guidance and for providing access to the FPGA development facilities used in this work.

REFERENCES

[1] E. Cheshmikhani, M. Saadatmand, M. Ekberg, and M. Sjödin, "Interrupt Caching: A Hardware-Assisted Interrupt Handling to Enhance System Responsiveness," IEEE Access, vol. 13, pp. 8842–8857, Jan. 2025.

[2] "FPGA Implementation of an Improved Watchdog Timer for Safety-Critical Applications," Int. J. Adv. Res. Eng. Technol. (IJARET), vol. 16, no. 11, Nov. 2025.

[3] R. Tamilselvan, "Design of the Configurable Watchdog Timer Using FSM Technique," Int. J. Sci. Res. Eng. Dev. (IJSRED), vol. 4, no. 3, pp. 378–384, 2021.

[4] C. D. J. Chaithanya and M. Thouqeer, "Microarchitecture and Design of a Watchdog Timer for a RISC-V Based SoC," in Proc. IEEE Int. Conf. Innov. Data Comput. Admin. (ICIDCA), 2023.

[5] J. Henkel and R. Ernst, "A Hardware–Software Approach for Fault Tolerance in Embedded Systems," eScholarship, Univ. of California, 2002.

[6] R. Tamilselvan, "FSM-Based Fault Recovery for Real-Time Embedded Systems," Int. J. Sci. Res. Eng. Dev. (IJSRED), 2021.

[7] S. G. Karthick and V. Ramalingam, "Efficient Reset Design Techniques in VLSI Circuits," J. Eng. Sci. Appl. (JESA), vol. 57, no. 6, 2022.

[8] K. T. Lisenbee, D. A. Ladd, and R. C. Baxter, "Watchdog Timer Circuit and Method for Monitoring Software Execution," U.S. Patent US20090119464A1, May 2009.