Mitigating Pipeline Hazards in RISC-V: Forwarding vs. Stalling in the Context of Continuous-Variable Quantum Key Distribution

David Machado Couto Bezerra[*,1], **Ramylla Luiza Barbalho Gomes Bezerra**[1], **Linton Thiago Costa Esteves**[1], **Nelson Alves Ferreira Neto**[1]

[1]*QuIIN - Quantum Industrial Innovation, EMBRAPII CIMATEC Competence Center in Quantum Technologies, SENAI CIMATEC, Salvador, BA, Brazil.*

[*]*Corresponding author: david.bezerra@fbter.org.br*

Abstract: The viability of continuous variable quantum key distribution (CV-QKD) systems critically depends on the performance of a computationally intensive digital signal processing (DSP) and post-processing sequence, including reconciliation, privacy amplification, and authentication. These operations require low-latency processing to sustain viable key rates, making the processor a key limiting factor in overall key generation. Application-specific instruction set processor (ASIP) accelerators offer a promising approach to address these limitations; however, their development relies on a base instruction set architecture (ISA). The open and modular nature of the fifth-generation Reduced Instruction Set Computing (RISC-V) standard makes it a strong alternative to proprietary designs, which often face customization and intellectual property barriers. This work presents an early-stage investigation into programmable accelerator microarchitectures, evolving from a single-cycle processor to a 5-stage pipeline. Both stall and forwarding mechanisms were implemented and compared to handle pipeline hazards. To validate the design, the central processing unit (CPU) was synthesized and deployed on a Xilinx Kintex-7 field programmable gate array (FPGA) using Vivado. Simulations revealed that, while both approaches were functionally equivalent and had similar hardware usage, the forwarding implementation executed in 25 cycles, a 57.6% reduction compared to the 59 cycles of the stalling version, resulting in shorter simulation time (255.0 ns vs. 590.0 ns). However, forwarding consumed more power due to extra control logic, whereas stalling delivered better timing performance and more robust margins under constraints. In conclusion, forwarding improves throughput, while stalling enhances timing robustness, guiding efficient RISC-V optimization for CV-QKD systems.
Keywords: RISC-V, Pipeline, Forwarding, Stalling, Continuous-Variable Quantum Key Distribution.

1. Introduction

Advances in quantum technologies, particularly CV-QKD, offer new possibilities for secure communication. However, the practical viability of these systems depends on a performance bottleneck: a computationally intensive post-processing sequence that demands low-latency processing to sustain viable key rates [1, 2]. This makes the processor a significant limiting factor in the overall key generation.

Overcoming this challenge requires dedicated hardware acceleration. The open and modular nature of the RISC-V architecture presents a robust alternative to proprietary designs [3], which are often hindered by customization restrictions. To maximize performance, pipelining is a fundamental technique [4], but it introduces challenges known as hazards [5] that must be managed. This work addresses these challenges through early-stage investigation by implementing and comparing techniques like forwarding and stalling [6].

This paper presents the development of a 5-stage pipelined RISC-V processor from a single-cycle design [7]. We detail the CPU's architecture, its support for a comprehensive subset of the RV32I instruction set, and the implemented hazard-handling mechanisms. The findings of this study offer a crucial contribution to the optimization of RISC-V processors, aiming to meet the

rigorous speed and performance requirements of high-demand applications like post-processing in CV-QKD systems.

The structure of this paper is organized as follows: Section 2 provides an overview of CV-QKD. Section 3 discusses the role of FPGAs in accelerating CV-QKD processing. Section 4 details the model and methods, including the RV32I architecture, pipelining strategy, and hazard types. Section 5 describes the design of the 5-stage pipelined RISC-V processor and its hazard mitigation units. Section 6 presents synthesis results, resource utilization, timing analysis, and power for both stalling and forwarding implementations. Section 7 describes the conclusion of the work and future research directions.

2. CV-QKD Systems

The core of CV-QKD systems is the secure distribution of symmetric cryptographic keys. This process relies on transmitting and detecting quantum states to create a raw key. However, the protocols require significant DSP operations.

A typical QKD system, including CV-QKD, follows main steps to establish a secret key:

Preparation, Distribution, and Measurement of Quantum States: During this phase, a sender (Alice) encodes keys into quantum states and sends them to a receiver (Bob) through a quantum channel. Bob then measures these received states using random or in both quadratures. In CV-QKD, the information is encoded into the amplitude and phase quadratures of the electric field in the optical phase space of the quantum states, which allows the use of high-sensitivity coherent detection techniques and DSP techniques to improve the information correlation.

Data Sifting: During the sifting phase, raw data corresponding to mismatched quadrature measurements is discarded. This process only takes place with random quadrature selection in homodyne detection.

Parameter Estimation: Subsequently, channel parameters (transmittance T and excess noise ξ) are estimated to bound the information accessible to an eavesdropper (Eve).

Information reconciliation (IR): This is one of the crucial steps in post-processing, where error-correcting codes are used to obtain identical keys.

Privacy Amplification (PA): Finally, it reduces leaked information through hash functions to generate shorter, information-theoretically secure secret keys.

3. Field-Programmable Gate Arrays

FPGAs have proven promising due to their superior parallel processing capabilities and lower power consumption compared to Graphics Processing Units (GPUs) [8, 9]. The challenge lies in optimizing the utilization of FPGA resources and effectively mitigating pipeline hazards within the

processor design to meet the rigorous speed and performance requirements of these systems.

In CV-QKD systems, the post-processing computation speed, which includes IR and PA, is a critical factor that affects the practical secret key rate [10]. By enabling the creation of specialized modules, FPGAs can perform these complex and computationally demanding DSP operations efficiently, thereby accelerating performance in terms of both distance and secret key rate.

4. Model and methods

Developing the RISC-V processor requires a clear methodological approach, which will be described in this section. To achieve the project's objectives, this work was divided into three main phases: (1) selecting and adapting the RV32I architecture; (2) applying pipelining to optimize instruction execution; and (3) implementing hazard mitigation strategies (stalling and forwarding).

4.1. RV32I Architecture

The RISC-V RV32I architecture, designed to be an efficient compiler target and to support modern operating systems [11], stands out for its base instruction set. This Instruction Set Architecture (ISA), which is standard in all RISC-V implementations, has a total of 47 instructions, as established in the manual [12]. In this work, we focus on the successful implementation of 37 of these instructions, Fig. 1, seeking a balance between functionality and minimal hardware requirements.

Figure 1: Implemented instructions.

ADD	ADDI	LH	BEQ	SLTI	BLTU
SUB	ANDI	LBU	BNE	SLTIU	BGEU
AND	ORI	LHU	BLT	SRAI	SRAI
XOR	XORI	SLLI	BGE	SW	JAL
SLT	LW	SRL	SRLI	SH	LUI
SLL	LB	SLTU	JALR	SB	AUIPC

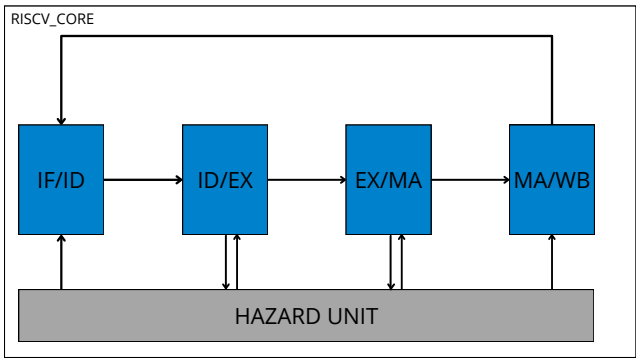
The execution of an instruction follows a well-defined flow: the program counter (PC) points to the address of the instruction to be fetched from instruction memory. The instruction is then decoded to identify the operation and the source registers. The necessary values are then read from the register file, if necessary. The arithmetic and logic unit (ALU) performs the operation, and the final result is either saved to data memory or written back to the register file, depending on the instruction type.

4.2. Pipelining

As shown in Fig. 2, the pipelining technique is classically divided into five stages: instruction fetch (IF), instruction decode (ID), execute (EX), memory access (MA) and write-back (WB). This approach offers substantial advantages in optimizing processor utilization and resolving performance bottlenecks [13, 14, 15]. By implementing overlapped instruction execution through pipeline registers between sub-processes, the instruction throughput [16] is improved, and the program ex-

ecution time is reduced, enhancing overall processor performance.

Figure 2: Architecture RISC-V Processor.



4.3. Hazards

When implementing pipelining, challenges known as hazards arise, which can compromise the efficient and correct execution of instructions. A hazard occurs when a dependency or resource conflict could cause an incorrect result during execution.

A pipeline can face three main types of hazards: structural hazards, which occur when two instructions try to use the same hardware resource at the same time; data hazards, which happen when an instruction needs the result of a prior instruction that hasn't finished yet; and control hazards, which arise with branch or jump instructions because the next instruction to fetch isn't known until the branch decision is made.

When forwarding is not feasible, the processor implements *stalling*, which inserts a delay cycle—commonly referred to as a "bubble"—into the pipeline to ensure that correct data is used. The strategic application of these two techniques optimizes processor performance, minimizes latency, and preserves the integrity of the RISC-V pipeline.

Stalling is particularly crucial for resolving load-use hazards. This occurs when an instruction attempts to use the value of a register that has just been updated by a load instruction.

The Fig.3 illustrates the execution flow of instructions within a RISC-V processor pipeline, specifically demonstrating the occurrence and resolution of a control hazard in a 5-Stage pipelined RISC-V processor. The hazard is caused by the branch instruction *beq x2, x6, 12*, whose decision is finalized only during the EX stage at clock cycle 6T. Prior to this, the subsequent instruction, *and x11, x3, x7*, is fetched speculatively in the IF stage at cycle 5T. To prevent the execution of a potentially incorrect instruction, the pipeline implements a stalling mechanism. This stall is explicitly shown by the insertion of no operation (NOP) instructions for the *and x11, x3, x7* instruction during clock cycles 6T, 7T, and 8T. This delay ensures that the pipeline waits for the branch outcome to be resolved before proceeding with the correct instruction flow, thereby maintaining program correctness.

5. RISC-V Processor Design

To implement a processor with both architectures, a rigorous design process is essential. As shown in Fig. 4, the design process begins with methodology definition. Next, SystemVerilog coding is

Figure 3: Stalling Example.

Clock Pulse/ → Instruction ↓	T	2T	3T	4T	5T	6T	7T	8T	N(T)	N+1(T)	N+2(T)	N+3(T)	N+4(T)
addi x1,x0,5	IF	ID	EX	MA	WB								
or x9,x8,x4		IF	ID	EX	MA	WB							
and x5,x3,x2			IF	ID	EX	MA	WB						
beq x2,x6,12				IF	ID	EX	MA	WB					
and x11,x3, x7					IF	NOP	NOP	NOP					
									IF	ID	EX	MA	WB

performed, followed by simulation for functional validation. If the design passes validation, it proceeds to integration. Otherwise, the process returns to the design initialization phase for optimizations and adjustments, creating an iterative refinement cycle until all requirements are met.

The five-stage pipelined processor shown in Fig.2 is designed to allow instruction operations to overlap, which significantly increases throughput and processing efficiency.The Hazard Unit, shown at the bottom of the diagram, is the crucial control logic for detecting and mitigating hazards in the pipeline. Its function is to ensure correct and timely instruction execution by applying mechanisms such as forwarding and stalling to resolve data and control conflicts.

From that groundwork, two distinct CPU implementations were developed, each adopting a different hazard-handling strategy. The first implementation uses a full stall policy, where pipeline execution halts entirely upon hazard detection, prioritizing design simplicity and predictability. The second implementation incorporates forwarding logic. A comparative analysis between the two CPU implementations will be conducted using the Vivado synthesis analysis tool, enabling informed assumptions regarding the impact of each hazard-handling strategy on performance, resource utilization, and design complexity.

6. Results

The design was synthesized and implemented on a Xilinx Kintex-7 FPGA (XC7K70T-FBV676-1) using Vivado. Resource utilization—particularly LUT and Flip-Flop (FF) consumption—was analyzed as a critical factor for embedded CV-QKD deployments, where cost-efficiency and power constraints are key considerations in hardware selection. A comparison between the stall and forward methods is shown in Figure 5.

Figure 4: Implementation flowchart.

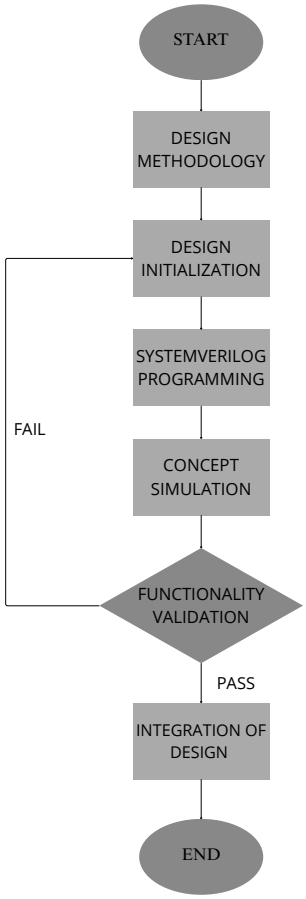


Figure 5: Percentage of hardware resources.

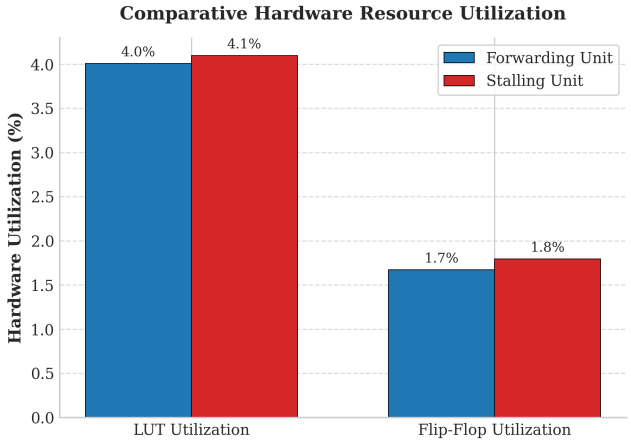
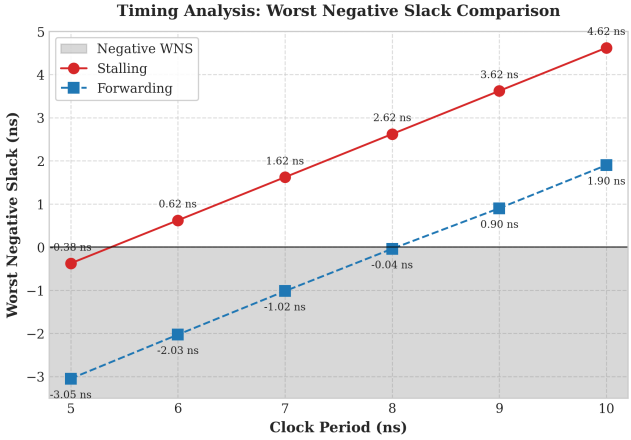


Figure 6: Contrast between different clock periods.



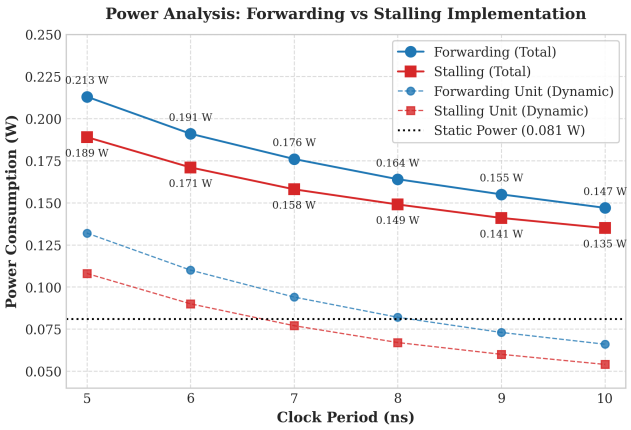
The results indicate that both architectures exhibit similar resource usage in terms of look-up tables (LUTs) and Flip-Flops, with the forward method demonstrating a slight advantage over the stall method. Figure 6 illustrates the worst negative slack (WNS) observed across varying clock periods for both the stalling and forwarding architectures, as obtained from static timing analysis (STA) reports.

As shown in Figure 6, the forwarding architecture exhibits a consistently negative slack at lower clock periods, indicating poorer timing performance under aggressive timing constraints. However, its slack improves linearly with relaxed clock periods, eventually meeting timing requirements at 8 ns and beyond. In contrast, the stalling architecture demonstrates superior timing characteristics throughout, achieving positive slack as early as 6 ns clock period and reaching slack of 4.62 ns at 10 ns. This suggests that, while forwarding may offer marginally better resource efficiency, stalling provides more robust timing margins under tighter constraints.

Figure 7 presents the power consumption comparison between forwarding and stalling architec-

tures across varying clock periods, based on post-implementation power analysis performed in Vivado. The results show that the forwarding implementation consistently consumes more power than the stalling counterpart across all evaluated clock periods. This is primarily due to increased dynamic power driven by the additional control logic inherent in forwarding.

Figure 7: Comparison between clock periods.



To evaluate functional behavior and cycle performance, both the stalling and forwarding architectures were simulated using the RISC-V code example provided in Harris's textbook, as previously referenced. The simulation results are summarized in Table 1. Both implementations produced the correct final value at memory address 100, confirming functional equivalence. However, the forwarding architecture completed execution in only 25 cycles, compared to 59 cycles for the stalling design, representing a 57.6% reduction in cycle count. Consequently, the total simulation time was significantly lower for the forwarding implementation (255.0 ns vs. 590.0 ns), clearly demonstrating its performance advantage due to reduced pipeline hazards and improved instruction throughput.

Table 1: Stalling vs Forwarding.

Metric	Stalling	Forwarding
Total Cycles	59	25
Simulation Time (ns)	590	255

As shown, the forwarding-based design significantly reduced execution cycles and simulation time, enhancing throughput, which is critical for the time-sensitive post-processing operations in CV-QKD. However, this gain came at the cost of increased power consumption and reduced timing margins under tight constraints. In contrast, the stalling approach showed better performance in timing analysis and lower dynamic power usage, making it a more stable choice for resource-constrained environments.

7. Conclusion

This work explored the development and evaluation of a pipelined RISC-V processor aimed at supporting CV-QKD systems. By implementing and comparing two key hazard mitigation strategies: stalling and forwarding, the study demonstrated the trade-offs involved in optimizing performance, timing robustness, and power efficiency. The forwarding design boosted throughput at the cost of higher power consumption and reduced timing margins, making it preferable for a CV-QKD system ASIP. In contrast, the stalling approach was more power-efficient and stable, making it better for resource-constrained systems.

These findings lay the groundwork for future research focused on refining processor architectures for quantum communication applications, with the goal of achieving optimal trade-offs between speed, efficiency, and reliability.

Acknowledgements

This work was fully funded by the project *HW DSP: Development and Prototyping of Multicore SoC with Dedicated Accelerators and RISC-V DSP*, supported by QuIIN – Quantum Industrial Innovation, the EMBRAPII CIMATEC Competence Center in Quantum Technologies, with financial resources from the PPI IoT/Industry 4.0 of the MCTI, grant number 053/2023, signed with EMBRAPII.

References

[1] Faeq Hussain and Santanu Sarkar. Design and fpga implementation of five stage pipelined risc -v processor. *2024 IEEE 9th International Conference for Convergence in Technology (I2CT)*, pages 1–6, 2024.

[2] Yichen Zhang, Yiming Bian, Zhengyu Li, Song Yu, and Hong Guo. Continuous-variable quantum key distribution system: Past, present, and future. *Applied Physics Reviews*, 11(1):011318, 03 2024.

[3] Xinyi Wu, Lianye Liao, Xiaodong Fan, Ye Chen, Jinquan Huang, Minjie Liu, Zhiyu Tian, Tonglin Mu, Junran Guo, Bo Liu, and Shihai Sun. A flexible soc for quantum key distribution post-processing based on risc-v processor. *Quantum Science and Technology*, 10(3):035027, may 2025.

[4] Shen-Shen Yang, Zhen-Guo Lu, and Yong-Min Li. High-speed post-processing in continuous-variable quantum key distribution based on fpga implementation. *Journal of Lightwave Technology*, 38(15):3935–3941, 2020.

[5] I. Thanga Dharsni, Kirti S. Pande, and Manoj Kumar Panda. Optimized hazard free pipelined architecture block for rv32i risc-v processor. *2022 3rd International Conference on Smart Electronics and Communication (ICOSEC)*, pages 739–746, 2022.

[6] S.-S. Yang, Z.-L. Bai, X.-Y. Wang, and Y.-M. Li. Fpga-based implementation of size-adaptive privacy amplification in quantum key distribution. *IEEE Photonics Journal*, 9(6):7600308, Dec. 2017.

[7] Sarah L. Harris and David Money Harris. *Digital Design and Computer Architecture: RISC-V Edition*. Morgan Kaufmann/Elsevier, Cambridge, MA, 2022.

[8] Z. Bai, X. Wang, S. Yang, and Y. Li. High-efficiency gaussian key reconciliation in continuous variable quantum key distribution. *Science China Physics, Mechanics and Astronomy*, 59(1):614201, Jan. 2016.

[9] Z. Bai, S. Yang, and Y. Li. High-efficiency reconciliation for continuous variable quantum key distribution. *Japanese Journal of Applied Physics*, 56(4):044401, Apr. 2017.

[10] G. Van Assche, J. Cardinal, and N. J. Cerf. Reconciliation of a quantum-distributed gaussian key. *IEEE Transactions on Information Theory*, 50(2):394–400, Feb. 2004.

[11] David A Patterson and John L Hennessy. *Computer Organization and Design RISC-V Edition: The Hardware Software Interface*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2017.

[12] Andrew Waterman, Krste Asanović, and Others. The RISC-V Instruction Set Manual Volume I: User Level ISA Document Version 2.2. Technical report, CS Division, EECS Department, University of California, Berkeley, May 2017.

[13] Don Kurian Dennis, Ayushi Priyam, Sukhpreet Singh Virk, Sajal Agrawal, Tanuj Sharma, Arijit Mondal, and Kailash Chandra Ray. Single cycle risc-v micro architecture processor and its fpga prototype. In *2017 Seventh International Symposium on Embedded Computing and System Design (ISED)*, pages 1–5, Durgapur, India, 2017. IEEE.

[14] Wendi Zhang, Yonghui Zhang, and Kun Zhao. Design and verification of three-stage pipeline cpu based on risc-v architecture. In *5th Asian Conference on Artificial Intelligence Technology (ACAIT)*, 2021.

[15] P Singh, A Rai, A Rajput, P C Joshi, and A Prakash. Design and analysis of high speed risc processor using pipelining technique. In *2022 4th International Conference on Advances in Computing, Communication Control and Networking (ICAC3N)*, pages 1642–1645, Greater Noida, India, 2022.

[16] Shuzhou Sun, Rui Zhang, and Hui Ma. Efficient parallelism of post-quantum signature scheme sphincs. *IEEE Transactions on Parallel and Distributed Systems*, 31(11):2542–2555, 2020.