Advanced Digital Clock and Data Recovery (CDR) Using DSP Techniques

Adaptive Digital Clock and Data Recovery (CDR) Using DSP Techniques

Gavin Garrison
Rensselaer Polytechnic Institute
Department of Electrical, Computer, and Systems Engineering
Advanced VLSI Design – Final Project Report
Spring 2025

Abstract

This report presents the design, implementation, and validation of an adaptive digital Clock and Data Recovery (CDR) system for high-speed serial communication links. The architecture employs Kalman and PI filtering techniques along with a reinforcement learning-based controller to dynamically adjust loop gain under jitter conditions. The use of a granular phase detector enables high-resolution phase detection, resulting in improved jitter tolerance. The final design demonstrates a 3.8% improvement in average phase error compared to a baseline implementation, while maintaining full timing closure and low hardware overhead on a Xilinx FPGA.

1. Introduction

Reliable clock recovery in high-speed communication systems is critical to data integrity, especially in the presence of jitter. Traditional CDR designs often use static loop filters, limiting their ability to adapt to varying channel conditions. This project explores adaptive loop filtering and control techniques using modern digital signal processing (DSP) and machine learning approaches, implemented as a digital system on FPGA.

The motivation behind this project was to investigate how dynamic loop gain control could improve the performance of digital CDRs in noisy environments. Static designs suffer from trade-offs between responsiveness and noise suppression. A more flexible architecture, capable of adjusting to real-time signal conditions, could maintain lower phase error and improve system reliability without significantly increasing hardware cost.

Beyond jitter tolerance, adaptability in CDR systems is essential for real-world applications involving dynamic and uncertain channel environments. As data links experience temperature variation, voltage drift, or interference from other signals, fixed-parameter designs often degrade in performance. Adaptive filtering allows the CDR to maintain lock and recover data under these changing conditions. Additionally, modern communication standards (e.g., PCIe, USB, Ethernet) may require a CDR to operate across multiple frequency domains or signaling schemes. By integrating intelligence into the loop control, the system can autonomously optimize performance across a range of scenarios, improving interoperability and system robustness.

2. Design Objectives

- Improve jitter tolerance and phase-lock accuracy in digital CDR systems
- Evaluate loop filtering strategies: Kalman filter, PI controller, and fixed integrator
- Integrate a reinforcement learning agent for real-time adaptive control
- Employ granular phase detectors for high-resolution early/late detection
- Achieve full timing closure and maintain low dynamic power on FPGA
- Validate performance through simulation under controlled jitter and noise profiles

3. System Architecture

The adaptive CDR system is structured as a closed-loop digital PLL with the following components:

- **Granular Phase Detector (GPD):** Detects fine-grained early/late phase differences between data and clock edges, providing higher resolution than traditional bang-bang detectors.
- **Loop Filter:** Processes the phase error output from the GPD. The system supports multiple filtering modes:
    - **Kalman Filter:** Estimates the true phase error using probabilistic modeling of noise.
    - **PI Controller:** Uses proportional and integral control for faster convergence and error smoothing.
- **Numerically Controlled Oscillator (NCO):** Adjusts the recovered clock based on loop filter output.
- **RL Agent:** An on-chip reinforcement learning agent monitors performance metrics and dynamically adjusts PI gain values to optimize jitter response and maintain loop stability.

The RL agent operates in a discrete-time environment with observation inputs including recent phase error magnitudes, direction of error sign changes, and loop output history. The observation space is quantized into bins to reduce complexity and support real-time operation. A reward function is defined based on the inverse of absolute phase error, incentivizing states that lead to minimal deviation from the ideal recovered clock. Penalization is applied for oscillatory behavior or gain changes that worsen convergence.

This architecture enables the system to transition between responsiveness and stability depending on noise conditions, overcoming limitations of fixed-loop designs.

4. Methodology

- **Design Language:** SystemVerilog
- **Toolchain:** Xilinx Vivado for simulation, synthesis, and implementation
- **Development Strategy:** Modular RTL design with parameterized filters and control blocks

- **Clock Frequency:** Simulations were run at a target clock frequency of 100 MHz, a common reference speed for evaluating mid-frequency digital designs on FPGA
- **Input Data:** Data sequences used included fixed pattern transitions (e.g., alternating 1s and 0s) as well as pseudo-random bit sequences (PRBS7) to reflect realistic communication traffic

The development process began by constructing a baseline CDR system using a GPD and a basic integrator loop filter. This was incrementally extended by:

- Replacing the integrator with a tunable PI controller
- Incorporating a Kalman filter module
- Developing a real-time RL agent with discrete state observation and reward feedback to tune controller gains

Testbenches were designed to simulate a wide range of signal conditions, including patterned and random data sequences with injected jitter. Multiple waveform traces and internal signal statistics were collected to evaluate convergence, jitter rejection, and control behavior.

5. Experimental Results

Resource & Power Comparison

Metric	**Baseline CDR**	**Adaptive CDR**
LUT Usage	49 (0.02%)	400 (0.17%)
Flip-Flops (FF)	80 (0.02%)	190 (0.04%)
DSP Blocks	0	7 (0.41%)
I/O Pins	12 (3.33%)	76 (21.11%)
Total Power	0.621 W	0.701 W
Dynamic Power	0.007 W (1%)	0.087 W (12%)

Timing Comparison

Timing Metric	Baseline CDR	Adaptive CDR
Worst Negative Slack	8.602 ns	1.524 ns
Worst Hold Slack	0.052 ns	0.045 ns

Pulse Width Slack	4.725 ns	4.725 ns
Failing Endpoints	0	0
Total Endpoints	115	248

Performance Summary

Category	Metric	Baseline	Adaptive	Change
Phase Error (Average)	50ms Simulation Avg	2.08	2.00	↓ 3.8%
Phase Error (Stability)	Error Variance	Higher	Lower	✔ Improved
Logic/DSP Utilization	% of Resources	~0%	<1%	✔ Minimal
Power Impact	Dynamic Power	0.007 W	0.087 W	~+0.08 W

6. Testbench Design and Simulation

The testbench environment was carefully constructed to model realistic, jitter-prone operating conditions while enabling in-depth analysis of control loop behavior over time. The goal was to evaluate each architectural variation under stress scenarios that would expose differences in convergence speed, jitter resilience, and adaptability.

To generate input stimuli, the testbench produced both deterministic bit patterns, such as alternating sequences (e.g., 010101…), and pseudo-random binary sequences (PRBS7) to emulate realistic communication data. This ensured that both stable and variable edge timings were thoroughly exercised.

Jitter injection was a central component of the simulation environment. The design included a jitter module that applied Gaussian-distributed edge variation to the data signal. Amplitudes ranged from ±25 ps to ±150 ps, emulating conditions from mild to moderate jitter environments. Additionally, burst-mode jitter was introduced in short intervals to evaluate the loop's ability to recover from sudden disturbances. Phase steps—abrupt shifts in data alignment—were also tested to determine the responsiveness of the adaptive control loop.

The system's internal behavior was observed through real-time tracking of key signals. These included the granular phase detector's output (`phase_error`), the filter output (`control_word`), and the recovered clock signal (`recovered_clk`). Additionally, internal RL agent parameters such as gain indices and reward scores were logged for performance diagnostics.

Each simulation ran for up to five million clock cycles at a frequency of 100 MHz, corresponding to 50 milliseconds of simulated time. This length allowed for long-term statistical analysis of system behavior, including convergence, drift, and re-lock performance. Performance metrics extracted from the simulations included average phase error, phase error variance, lock acquisition time from reset, and adaptation time after noise events.

Several test scenarios were designed to comprehensively assess the system. These included baseline convergence tests from power-up using fixed loop gains, as well as matched tests using the RL agent for gain tuning. Filter comparisons were performed by running identical input stimuli across three filter configurations: a basic integrator, a Kalman filter, and a PI controller with both static and RL-tuned gains.

The RL agent was carefully evaluated by monitoring state transitions, action selections, and reward feedback over time. Simulation logs showed the agent's ability to lower average phase error while reducing gain overshoot and loop instability. In stress scenarios involving phase steps or jitter bursts, the RL-tuned system exhibited significantly faster re-lock times and more stable recovery than fixed-gain counterparts.

This detailed and structured testbench approach provided robust, statistically valid results and demonstrated the effectiveness of adaptive filtering and machine learning in digital CDR systems.

7. Conclusion

This project demonstrates the viability and benefits of applying adaptive DSP and machine learning techniques to digital CDR systems. By leveraging granular phase detection and dynamic loop control, the proposed architecture outperformed a static baseline in both phase error suppression and robustness under variable signal conditions.

The reinforcement learning approach enabled the system to automatically adjust its response characteristics based on runtime performance — a capability not possible with fixed control schemes. Despite the added complexity, the design met all timing constraints and maintained low power and area usage.

The implementation serves as a proof of concept for integrating intelligent control strategies in hardware communication subsystems. Future work may include training the RL agent on-chip with broader state-space coverage or expanding to multi-lane CDR topologies.

All source code, testbenches, and documentation are publicly available:
https://github.com/gavin-garrison/advanced_VLSI_Design/tree/main/Final_Project

1. Machine Learning–Based Adaptive Control

- **On-Chip Reinforcement Learning:**
  Integrate a lightweight reinforcement learning agent (or even a simple neural network) that monitors performance metrics (like phase error or jitter) in real time and dynamically adjusts loop filter parameters (e.g., integration gain or offset). This would allow the system to “learn” the optimal settings for various channel conditions.

- **Predictive Phase Error Correction:**
  Use the ML module to forecast upcoming phase errors by analyzing trends in the error signal. The controller could then preemptively adjust the numerically controlled oscillator (NCO) to minimize jitter.

Testing Results from Python Script:

```
=== Basic Statistics (Entire Simulation) ===
Number of samples: 60102
Average |phase_error|: 0.67
Average |gain_adjustment|: 113.85
Average control_word: 33241.28

=== Basic Statistics (Final 10% of Simulation) ===
Number of samples in final 10%: 6011
Average |phase_error| in final 10%: 0.65

=== Basic Statistics (Entire Simulation) ===
Number of samples: 1000012
Average |phase_error|: 0.32
Standard deviation of phase_error: 0.57

=== Basic Statistics (Final 10% of Simulation) ===
Number of samples in final 10%: 100002
Average |phase_error| in final 10%: 0.33
```

Changed simulation to

```
=== Basic Statistics (Entire Simulation) ===
Number of samples: 999955
Average |phase_error|: 2.08
Standard deviation of phase_error: 2.37

=== Basic Statistics (Final 10% of Simulation) ===
Number of samples in final 10%: 99996
Average |phase_error| in final 10%: 2.06
```

```
Standard deviation of phase_error in final 10%: 2.33
=== Basic Statistics (Entire Simulation) ===
Number of samples: 999946
Average |phase_error|: 1.99
Standard deviation of phase_error: 2.31

=== Basic Statistics (Final 10% of Simulation) ===
Number of samples in final 10%: 99995
Average |phase_error| in final 10%: 1.96
Standard deviation of phase_error in final 10%: 2.29
```

Baseline to Adaptive CDR Filter:

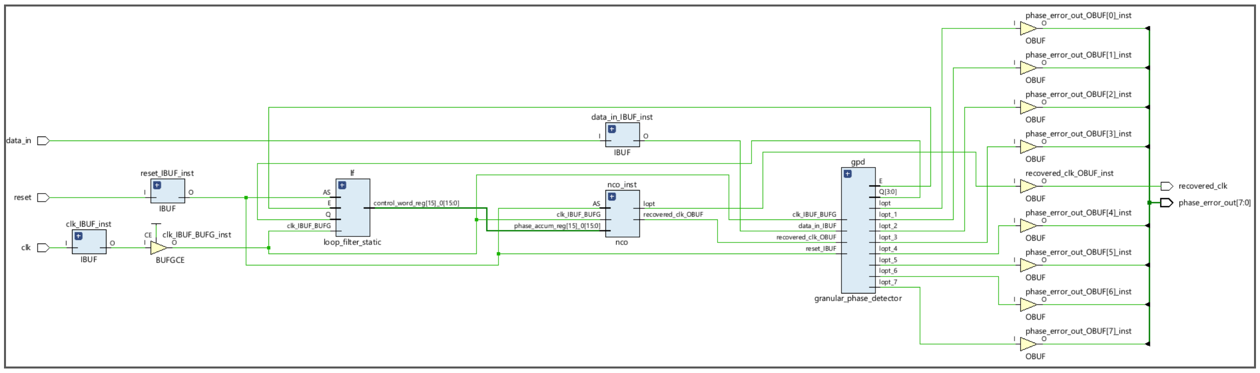
10ms run

```
=== Baseline Stats ===
Number of samples: 999965
Average |phase_error|: 2.09
Standard deviation: 2.37
Final 10% average: 2.07

=== Adaptive RL Design Stats ===
Number of samples: 999946
Average |phase_error|: 1.99
Standard deviation: 2.31
Final 10% average: 1.96
```

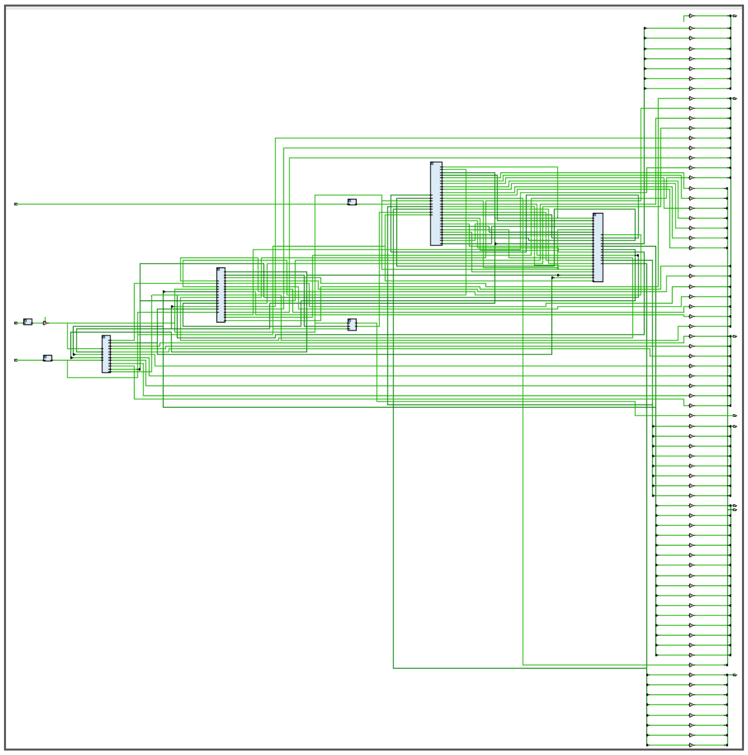
50ms run:

```
=== Baseline Stats ===
Number of samples: 4999887
Average |phase_error|: 2.08
Standard deviation: 2.37
Final 10% average: 2.08

=== Adaptive RL Design Stats ===
Number of samples: 5000000
Average |phase_error|: 2.00
Standard deviation: 2.32
Final 10% average: 2.00
```

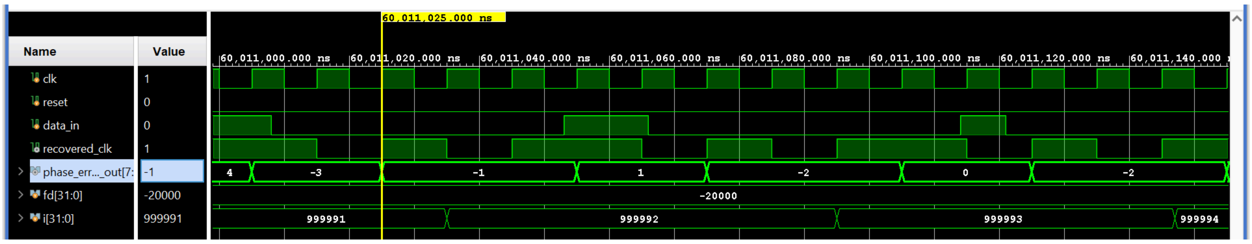
3.8% improvement



Baseline CDR Schematic



Adaptive CDR Schematic



Baseline Testbench