Design eines Redis Cache

Uni-Projekt: Chip Design & Verilog

Philipp Hecht Luca Pinnekamp Luca Schmid

27. Februar 2026

Zusammenfassung

Implement a Redis-inspired cache that works in conjunction with the CPU. The goal is to create a fast cache for storing key-value pairs. Basic CRUD (Create, Read, Update, Delete) implementation

Inhaltsverzeichnis

1 Einleitung / Idee

Die ursprüngliche Idee dieses Projekts ist der Entwurf und die Implementierung eines kompakten, synthetisierbaren Key-Value-Stores, inspiriert von Redis, auf RTL-Ebene (für FPGAs oder ASICs).

Das Ziel ist es, grundlegende Speicheroperationen direkt in Hardware abzubilden, um eine hohe Performance und geringe Latenz zu erreichen. Die ursprünglich geplanten Kernfunktionen sind:

- **Einfügen von Schlüssel-Wert Paaren (Key-Value Insertion)**
- **Abrufen von Werten anhand von Schlüsseln (Value Retrieval)**
- **Löschen von Werten anhand von Schlüsseln (Key Deletion)**
- **Auflisten von Schlüsseln (Key Listing)**
- **Automatische Ablaufzeit (TTL - Time-to-Live)**

Die Motivation liegt darin, die Effizienz von Key-Value-Speichern durch Hardwarebeschleunigung zu untersuchen und eine Schnittstelle bereitzustellen, die ähnlich wie Software-Caches funktioniert, aber die Vorteile dedizierter Hardware nutzt.

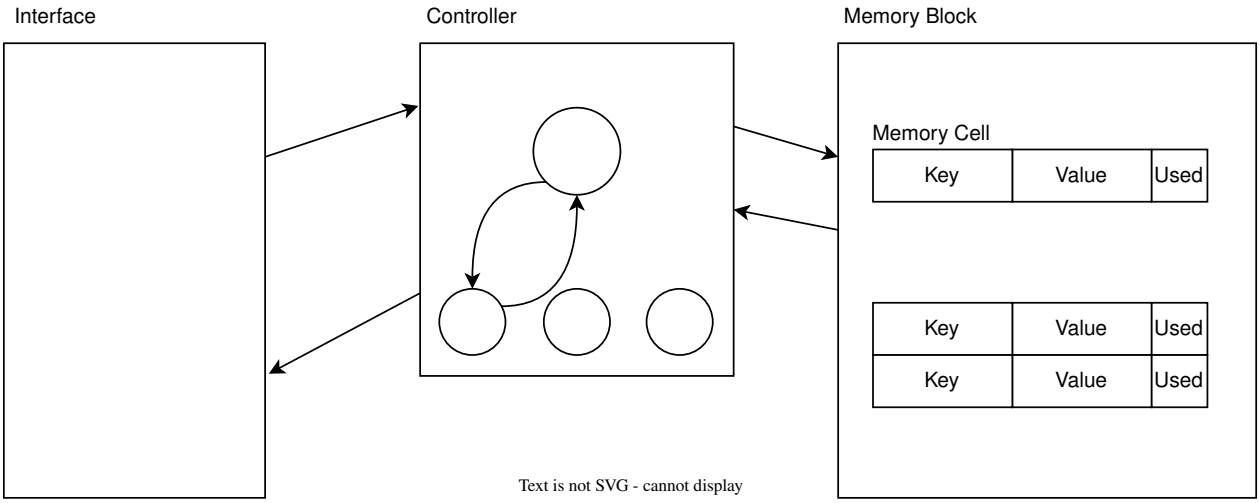

Abbildung 1: Speicherblöcke

2 Projekt Setup

2.1 Repo Struktur

2.2 Pipelines

3 Architektur

Notiz: Key = 0 bedeutet, dass die Speicherzelle nicht belegt ist (siehe memory_cell)

3.1 Alles richtung Architektur

3.2 Statemachine

Ursprünglich wurde GET, UPSERT und DELETE mit mehreren States designed. Bei der Implementation ist aufgekommen, dass alles mit einem Sate geht. Die Implementation für

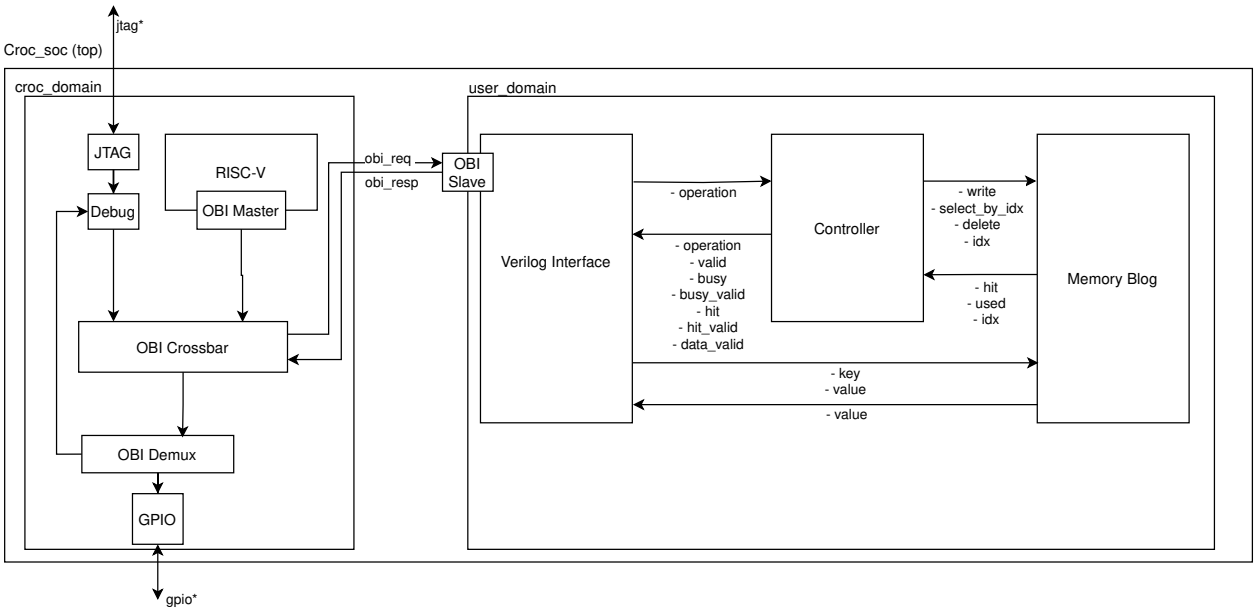

Abbildung 2: Architektur

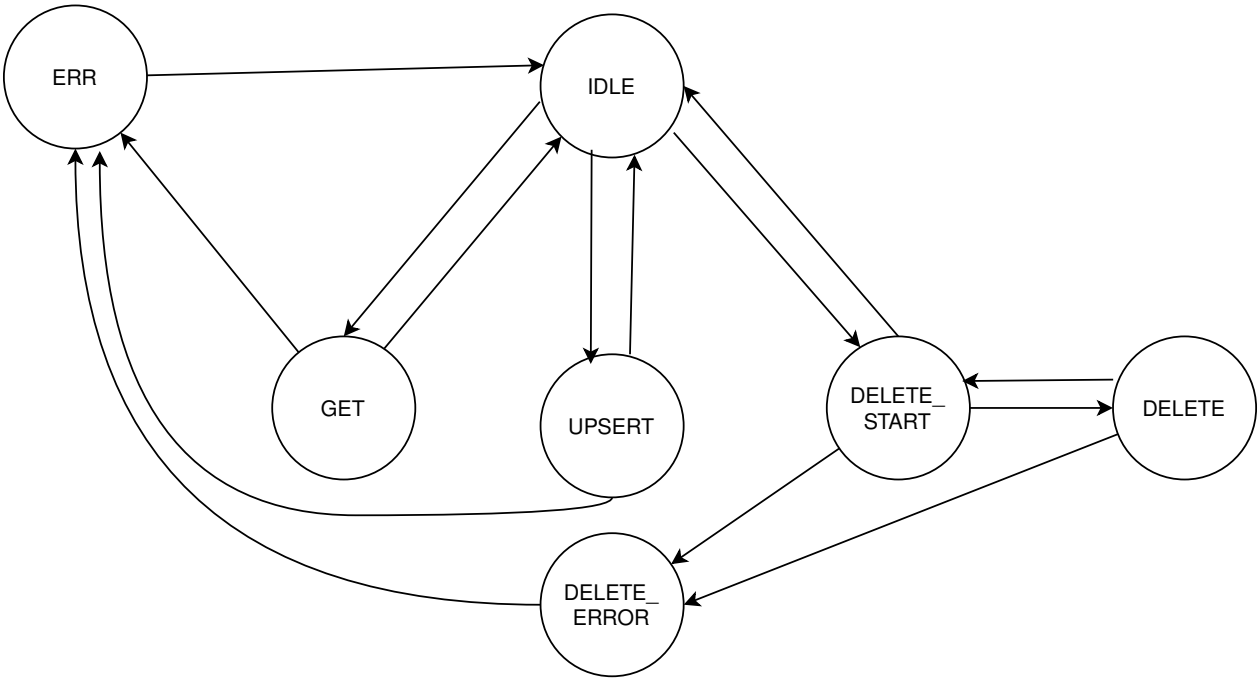

Abbildung 3: Statemachine

DELETE wurde aus Zeitrgründen nicht refactored. (Änderung der Implementation im Ausblick beschreiben oder zumindest in eine Liste eintragen)

Beispiel Code wie bei uns die Statemachine in die einzelnen Operationen unterteilt wurde.
cmd.done, cmd.error übergabe
en und enter

3.3 Taktzyklus Beispiele

Abbildung 4: taktzyklus

4 Implementationen

5 CROC

5.1 CROC Architektur

croc_soc
croc_domain
user_domain
alles andere

5.2 Theorie

5.3 Implementation bei uns im Projekt

6 OBI

6.1 OBI Protokoll

6.2 Register

Das OBI Protokoll wird genutzt um Daten in Register zu schreiben und daraus auszulesen. Standardmäßig sind folgende Register vorhanden:

	Value	Key	Ctrl
Größe	64 (8 Bytes)	32 (4 Bytes)	32 (4 Bytes)
Offset	0	8	12

Die Größe der Register ist in der cache_cfg_pkg. Die Offset Werte leiten sich aus den den Größen der Key und Value Register ab. In den OBI Nachrichten wird das Offset (addr) mitgegeben. Anhand des Offsets können Daten in das richtige Register geschrieben werden.

6.2.1 OBI Request

Die OBI Request wird von dem Master an den Slave (Interface) verserndet. Die 72 Bit lange Nachricht setzt sich aus dem Adress-Channel und dem Kontrollsignal zusammen.

Feld in obi_req_t	Bit-Breite	Beschreibung
addr	32	Register Speicheradresse: Der Wert entspricht dem Offset um die übermittelten Daten in dem richtige Register zuzuweisen
we	1	Write Enable: 1 bedeutet Schreiben, 0 bedeutet Lesen.
be	4	Byte Enable: Gibt an, welche Bytes der 32-Bit-Daten (wdata) gültig sind. Für ein volles 32-Bit-Wort ist das 1111
wdata	32	Write Data: Die Daten, die in den Speicher/das Register geschrieben werden sollen.
aid	1	Address ID: Eine ID für die Transaktion
a	1	Optional: Ein optionales Signal des OBI-Standards (in der Minimal-Konfiguration 1 Bit groß).
req	1	Request: Das Handshake-Signal. Wenn 1, bittet der Master um eine Transaktion.

6.3 OBI Response

Die OBI Response wird von dem Slave (Interface) an den Master versendet. Die 37 Bit lange Nachricht setzt sich aus dem R-Channel, dem Grant un dem Valid Signalen zusammen.

Feld in obi_rsp_t	Bit-Breite	Beschreibung
rdata	32	Read Data: Die Daten, die vom Interface gelesen wurden
rid	1	Response ID: Spiegelt die aid aus dem Request wider, um Antworten zuzuordnen
err	1	Error: Wird 1, falls beim Zugriff ein Fehler aufgetreten ist (z. B. falsche Adresse).
r	1	Optional: Ein optionales Signal für die Response (in der Minimal-Konfiguration 1 Bit).

Feld in obi_rsp_t	Bit-Breite	Beschreibung
gnt	1	Grant: Handshake-Signal. Der Slave setzt dieses Bit auf 1, um zu signalisieren: “Ich habe deinen Request (req=1) akzeptiert und verarbeite ihn”
rvalid	1	Response Valid: Wird 1, wenn die zurückgegebenen Daten in r.rdata gültig sind.

7 FPGA

8 RISCV

9 Backend

10 Learnings

11 Ausblick / Zusätzliche Funktionalitäten

- DELETE Implementation refactoren. Kann auf 1 State gekürzt werden

12 Vivado Setup (Mac Anleitung)