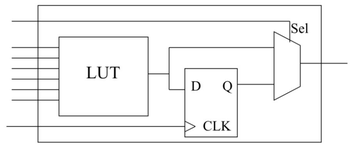
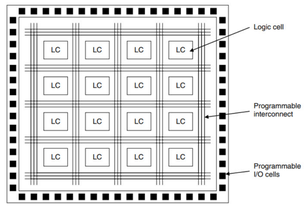
Die Simulation eines Quantencomputers auf einem FPGA

Einführung in FPGAs

Einfache Logikzelle

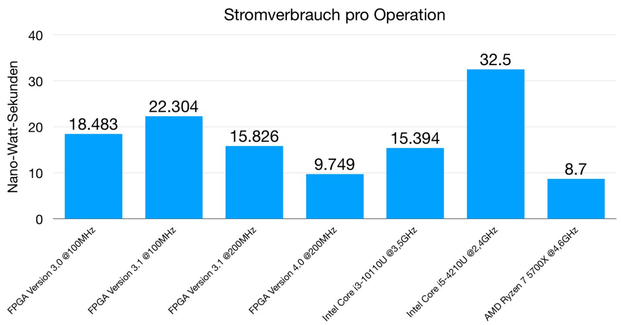


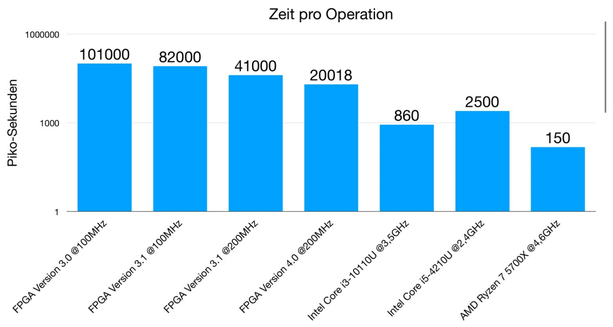
Zusammenschaltung vieler Logikzellen



- FPGA steht für Field-Programmable Gate Array.
- Es handelt sich um integrierte Schaltungen, die auf hardwareebene Programmiert werden können.
- FPGAs bestehen aus einer Vielzahl von Logikzellen, die sich flexibel konfigurieren lassen.
- **Logikzellen** sind die Grundbausteine eines FPGAs und bestehen typischerweise aus einem **Look-Up Table (LUT)**, einem **Flip-Flop** und manchmal zusätzlichen Logikkomponenten.
- Ein **LUT** (Look-Up Table) implementiert eine Wahrheits-Tabelle, die die logischen Ausgänge für alle möglichen Kombinationen der Eingänge speichert, wodurch beliebige logische Funktionen
  realisiert werden können.
- Durch die **Rekonfigurierbarkeit** können FPGAs für verschiedene Anwendungen genutzt werden, ohne dass neue Hardware entwickelt werden muss.
- **In diesem Projekt wird ein FPGA verwendet, um die Simulation von Quantencomputern hardwareseitig zu beschleunigen.**

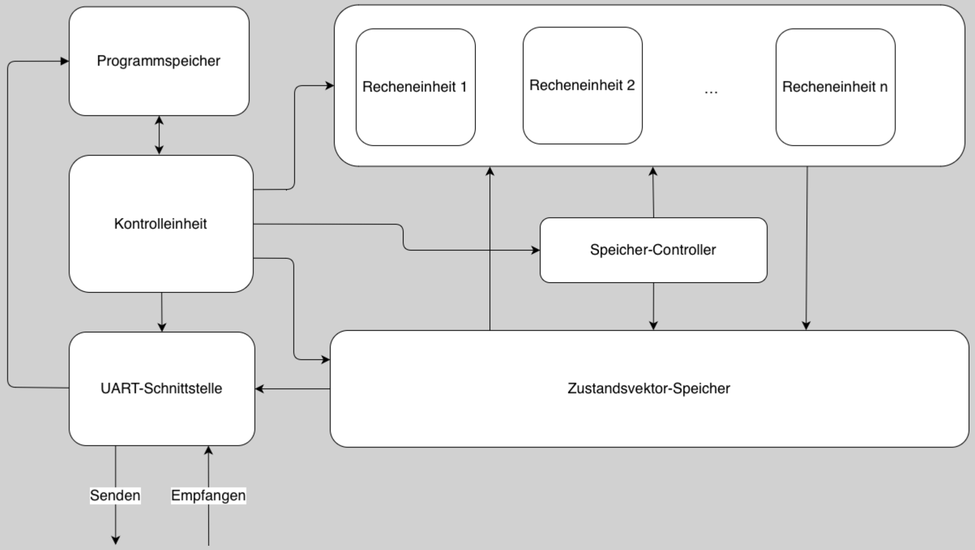
Ergebnisse und Performance





Alle Ergebnisse beziehen sich auf das Durchführen der Berechnungen ohne das Senden des finalen Zustandsvektors über die UART-Schnittstelle. Die Referenzwerte mit x86-Prozessoren wurden mithilfe der Python-Bibliothek **Qiskit** erzielt. Als „**Operation**“ wird die Aktualisierung eines Elements des Zustandsvektors bezeichnet. Die Daten zeigen Durchschnittswerte beim Simulieren von zufällig erstellten Quantenschaltkreisen.

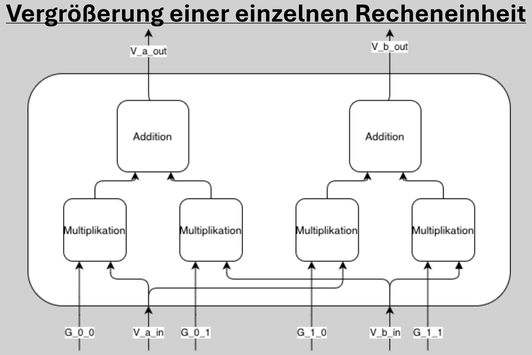
Architektur des Chips



Funktionsweise der Architektur bei der Durchführung eines Quantenalgorithmus

1. Quantenalgorithmus wird in Python definiert
2. Compiler erzeugt aus dem Python-Code eine für den FPGA lesbare Binärdatei
3. Binärdatei wird über die UART-Schnittstelle in den Programmspeicher des FPGAs geladen
4. Simulation wird gestartet: Kontrolleinheit liest die Instruktionen aus dem Programmspeicher
5. Ausführung einer Matrix-Multiplikation mit dem Zustandsvektor:
   1. Kontrolleinheit stellt allen Recheneinheiten die anzuwendene 2x2 Matrix zur Verfügung (G_0_0 bis G_1_1)
   2. Die Aktualisierung des Zustandsvektors erfolgt in Paaren aus jeweils zwei Elementen:
      1. Kontrolleinheit errechnet Indizes der Paare und gibt Informationen an den Speicher-Controller
      2. Speichercontroller lädt die jeweiligen Elemente aus Zustandsvektor-Speicher in verfügbare Recheneinheiten und nach der Aktualisierung zurück in den Speicher
   3. Wenn alle Paare aktualisiert wurden, liest die Kontrolleinheit die nächste Instruktion aus dem Programm-Speicher
6. Sobald alle Instruktionen ausgeführt wurden, werden die Elemente des finalen Zustandsvektors über die UART-Schnittstelle zurückgegeben und können so ausgelesen werden

Vergrößerung einer einzelnen Recheneinheit



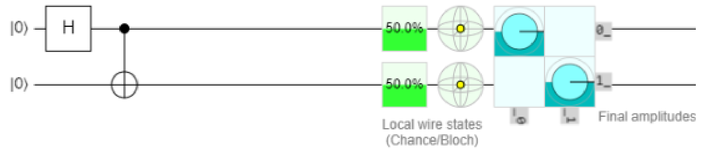
Compiler und Interface

Code zum Erstellen und kompilieren eines simplen Quantenschaltkreises in Python

```python
#create a simple quantum circuit with two qubits
circuit = QuantumCircuit(nQubits=2)
circuit.h(target=0)
circuit.cnot(target=1, control=0)

#create the compiler object and compile the circuit
compiler = FPGAQCCompiler()
compiler.compile(circuit=circuit, filepath=FPGAProgramsPath, filename="TestProgram")
```

Der gleiche Schaltkreis visualisiert und simuliert mit **Quirk**:



Kompilierte Binärdatei des Schaltkreises, die auf den FPGA hochgeladen werden kann:

```
00000000 --tut nichts
11000000 --setze Timer zurück
11000001 --starte Timer
00010010 --Setze Anzahl der Qubits auf 2
00100000 --Setze Ziel-Qubit auf 0
00110001 --Setze Ziel Matrix auf Hadamard
01000000 --Berechne Zustandsvektor
00100001 --Setze Ziel Qubit auf 1
00110011 --Setze Ziel Matrix auf X (in diesem Fall entspricht dies dem CNOT-GATE)
11010000 --Setze Kontroll-Qubit auf 0
01000000 --Berechne Zustandsvektor
11000010 --stoppe Timer
10000000 --Setze Adress-Register auf 0
01110000 --sende ein Element des Zustandsvektors über UART
10000010 --inkrementiere Adress-Register
10010011 --Gehe zurück zu Zeile 14 bis alle Elemente übertragen wurden
10110000 --Halt
```

Ausblick

Aktuelle Probleme der Architektur:

- Die maximale Simulationsgröße ist aufgrund des limitierten Speichers auf 14 Qubits begrenzt.
- Die Speicherbandbreite ist nicht groß genug, um mehr als 2 Recheneinheiten auszulasten.
- Die UART-Schnittstelle ist zu langsam, um die Ergebnisse effizient zu übertragen.

Mögliche Lösungen in der Zukunft:

- Effizienteres Pipelining, um mehr Recheneinheiten auszulasten und Performance zu erhöhen.
- Benutzung von externem Speicher, um größere Simulationen zu ermöglichen.
- Anbindung über PCIe, um ggf. Zugriff auf externen Arbeitsspeicher zu bekommen und eine effizientere Übertragung der Ergebnisse zu ermöglichen.