Thema:

Verbesserung eines Verifikationsworkflows für FPGA-Designs zur Anwendung

bei Quantencomputern

Kurzbeschreibung:

Problemstellung und Ziel:

In Quantencomputern erfordert die kurze Kohärenzzeit von Qubits schnelle

Mess- und Steuerungsmöglichkeiten. Existierende Netzwerkswitches und Hubs

genügen den spezifischen Anforderungen eines Quantencomputers nicht, so

dass spezialisierte Lösungen entwickelt werden müssen. Eine Möglichkeit

zur Implementierung dieser sind FPGA-basierte Chips, die über

Highspeed-Transceiver kommunizieren. Bei der Entwicklung von komplexen

Hardwarelösungen ist die funktionale Verifikation dieser ein wesentlicher

Bestandteil des Entwicklungsprozesses. Die sogenannte Simulation erfolgt

über Analyse und Kompilierung der in HDL spezifizierten Elemente in ein

Simulationsmodell welches zur Ausführung in eine Simulationsengine kommt.

Ziel der Arbeit war es diesen komplexen und aufwendigen Prozess zu

analysieren, die Möglichkeiten für eine Verbesserung auszuloten und

möglichst zu implementieren.

Geplantes Vorgehen:

Erste Schritte werden darin bestehen, die Funktionsweise eines

Quantencomputers und deren Abhängigkeit von schneller Steuerung und

Messung zu erläutern. Dann soll auf die Vorteile eines FPGAs im Kontext

eines solch forschungsträchtigen Gebiets eingegangen werden. Daraufhin

will ich auf die Bedeutung einer schnellen Simulation für die Verifikation

von Hardwaredesigns eingehen. Dann wird es Zeit den bisherigen Buildprozess

- bestehend aus Analyse und Synthese in Xilinx Vivado und Kompilierung und

Ausführung in Cadence NCSim -darzustellen. Infolge sollen die

Möglichkeiten für eine Beschleunigung, Automatisierung und Erweiterung

dieses Vorgangs gezeigt werden. Abschließend will ich den Mehrwert, aber

auch Schwierigkeiten oder Grenzen einer Implementierung jener erklären.

1. Abstract: Summary
2. Introduction: Context, problem and technological options
   1. Race for quantum supremacy - March '18: Google takes over IBM's 50 qubit processer  
      Briefly: how can fast simulation help to ensure competition?
   2. Context: Quantum computers, FPGAs and functional verification
      1. Quantum computers  
         How does a quantum computer work roughly? What are qubits? What are possible applications of quantum computing?
         1. Qubits, superposition and fundamentals of quantum mechanics
         2. Quantum operations and the essence of quantum supremacy
         3. Applications of quantum computing: Grover and Shor
         4. The challenge of quantum decoherence
            1. Error correction
            2. Architectural requirements

Cooling and isolation

FPGAs as reconfigurable ICs  
What is an FPGA? What are its benefits compared to other processors? What are HDLs? Why use an FPGA in this application?

Fast sample and control – high-speed serial communication

* + 1. Functional verification and its tools  
       What role play functional verification in FPGA design? Why does it depend on fast and easy simulation? What other forms of verification are there?  
       What are HDL simulators? How do they work in building a testable simulation?
  1. Problem: Complex and state-independent multi-tool build workflow
     1. Analysis and synthesis in Xilinx Vivado
     2. Compilation and simulation in Cadence Incisive  
        Roughly why were the previous build process too complex to be completed manually? In which way was the process ignorant towards already completed builds?
  2. Technological solution: make
     1. Preconditions: bash, current state of Git  
        What was there to work with?
     2. Alternatives to Make - pros and cons  
        Maven, Ant, Gradle  
        Why was make a good solution?

1. Problem solution
   1. Analysis of build process  
      What were the exact steps to be completed in order to simulate the hub?  
      What were the dependencies? What was created by what?  
      What tools did I use to analyze the process?
      1. Build baseline and necessary steps
      2. File dependencies and corresponding commands
   2. Build implementation in a Makefile
      1. Build setup
         1. Inter-Process Communication via Named Pipes  
            Why was this the way to handle the problem?  
            In the end how was communication channeled?  
            How are Vivado errors handled?  
            Why did they have to be temporary?
         2. Vivado project initialization – order-only Prerequisites  
            How did we make sure the project was setup correctly?
         3. Clean-Up of build remainders with phony targets
      2. Representing compound build milestones with empty targets  
         How did we use it to indicate the project is set up correctly? What were other applications?
      3. Dependency mapping - Pattern rules versus generated recipes  
         Why couldn't patterns be used in some cases?
2. Summary and outlook
   1. Task at hand, approach and result (short and precise)  
      Benefits and boundaries of the Makefile
   2. Further possibilities of automation and acceleration  
      Why not implemented? Difficulties?