technische universität
dortmund

Beispielhafte Entwicklung einer Hardware-Produktlinie anhand einer UART-Familie

Jan Dworschak
Abschlussvortrag Bachelorarbeit

TU Dortmund

jan.dworschak@tu-dortmund.de

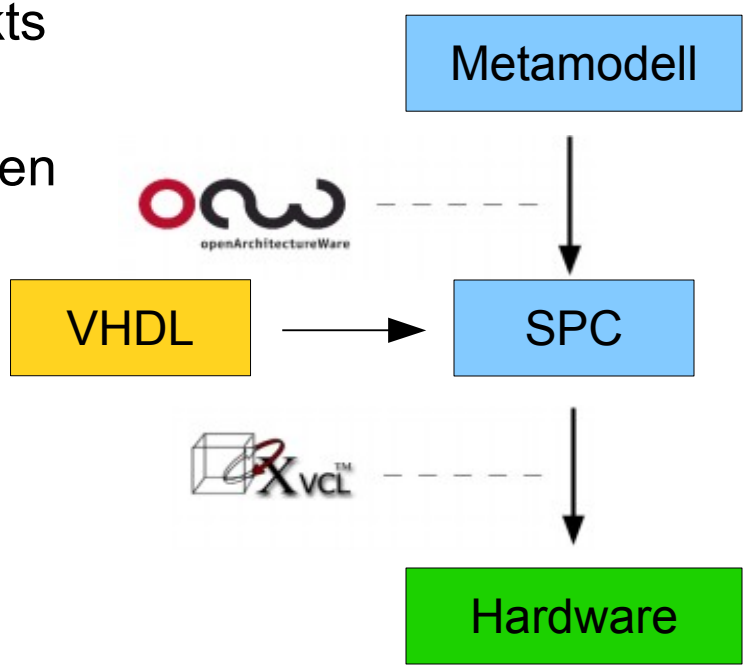
ESS

Übersicht

- Einführung
  - Kontext der Arbeit
  - Grundlagen
  - Ziele
- Entwurf des UART
- Implementation
  - VHDL-Komponenten
  - Codegenerierung
- Evaluation
  - UART
  - Codegenerierung
- Fazit

Einführung

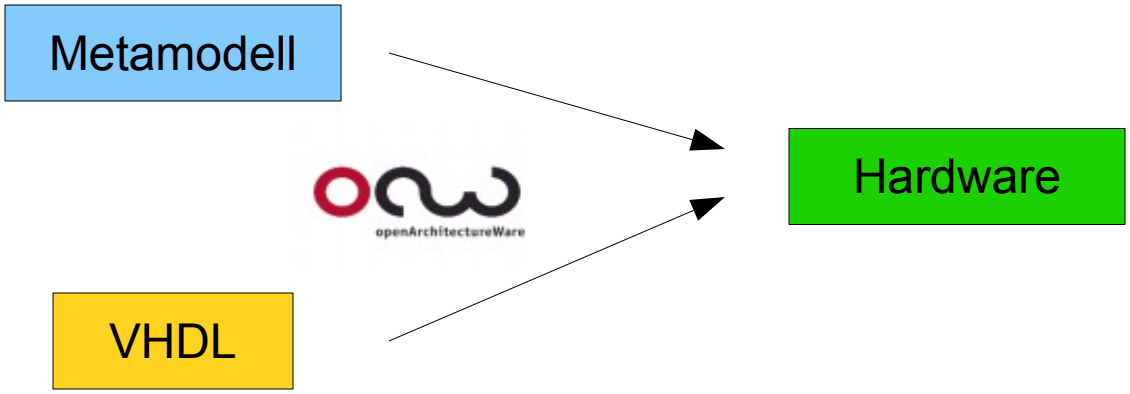
- Kontext
  - Hardwareebene des LavA-Projekts
  - Maßschneiderung von MPSoCs
  - Konzept der Hardwareproduktlinien



- Grundlagen
  - (LavA-)UART
    - Seriell/Parallel-Übersetzer
    - hier: RS-232/Wishbone
    - RX/TX mit einfachen Interrupts
  - LavA-Workflow
    - Modellierung und Konfiguration über Metamodell (Eclipse)
    - Erzeugung eines SPC für XVCL mittels oAW-Xpand
    - Codegenerierung über XVCL

Einführung

- Ziele der Arbeit
  - Erweiterung des statischen LavA-UART
    - konfigurierbare VHDL-Komponenten
    - C-Funktionen zur Ansteuerung und Laufzeitkonfiguration
  - Exemplarische Abwandlung des Workflows
    - Elimination von XVCL
    - Codegenerierung ausschließlich über Xpand-Framework

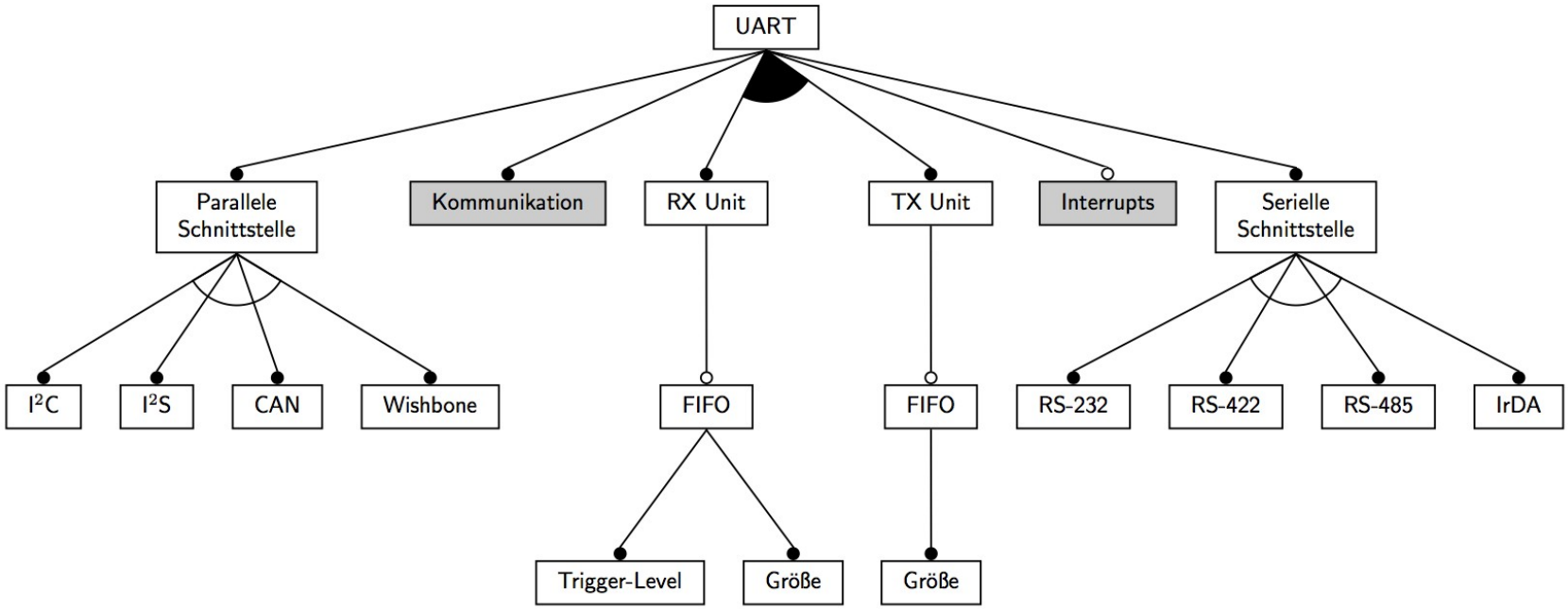


Übersicht

- Einführung
  - Kontext der Arbeit
  - Grundlagen
  - Ziele
- Entwurf des UART
- Implementation
  - VHDL-Komponenten
  - Codegenerierung
- Evaluation
  - UART
  - Codegenerierung
- Fazit

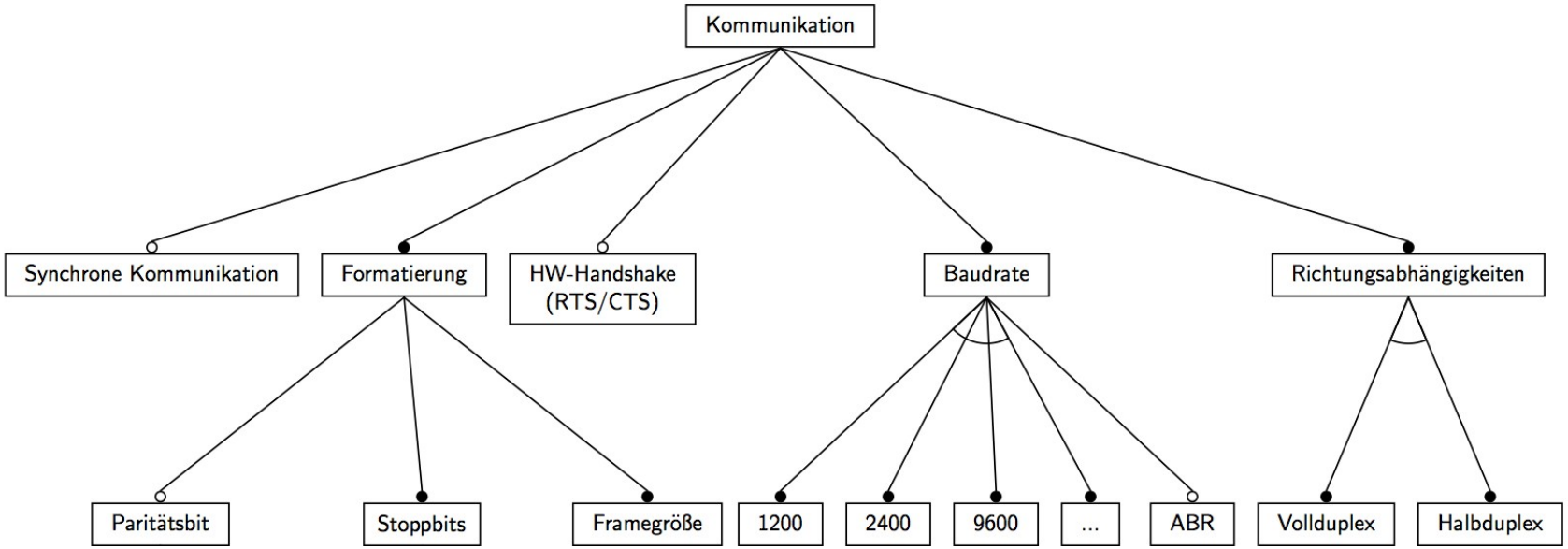
Entwurf des UART

- Domänenanalyse
  - Ziel: Abstecken eines sinnvollen Funktionsumfangs
  - Betrachtung verschiedener UART-Lösungen
    - Open Source VHDL-UARTs
    - UARTs gängiger Mikrocontroller



Entwurf des UART

- Kommunikationsbezogene Merkmale

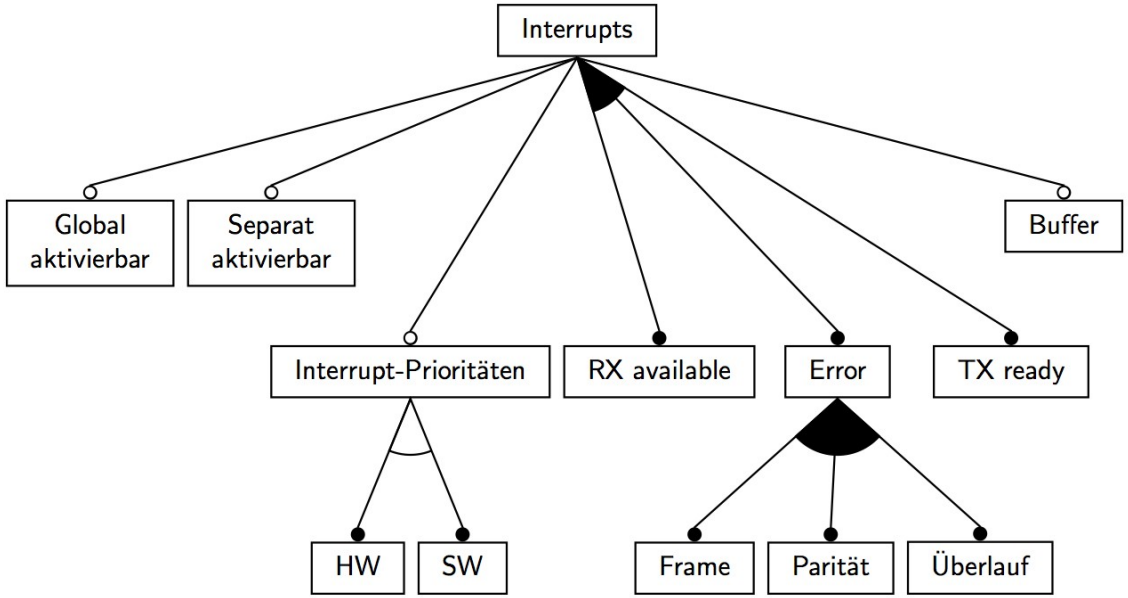


- Kommunikationsparameter:
  - Paritätsbit: keines/gerade/ungerade
  - Stoppbits: 1/1,5/2
  - Framegröße: 5 bis 9
  - Handshaking

Entwurf des UART

- Interruptbezogene Merkmale

- 6 Interrupts
  - Daten empfangen
  - Senden beendet
  - Framefehler
  - Paritätsfehler
  - Überlauf (RX/TX)



- Zusätzliche Features
  - Priorisierung von Interrupts
  - Pufferung
  - Konfiguration zur Laufzeit

Entwurf des UART

- Umsetzung der Konfigurationspunkte
  - Aufteilung nach Modellierungs- und Laufzeit
  - Kriterium: Ressourcenverbrauch

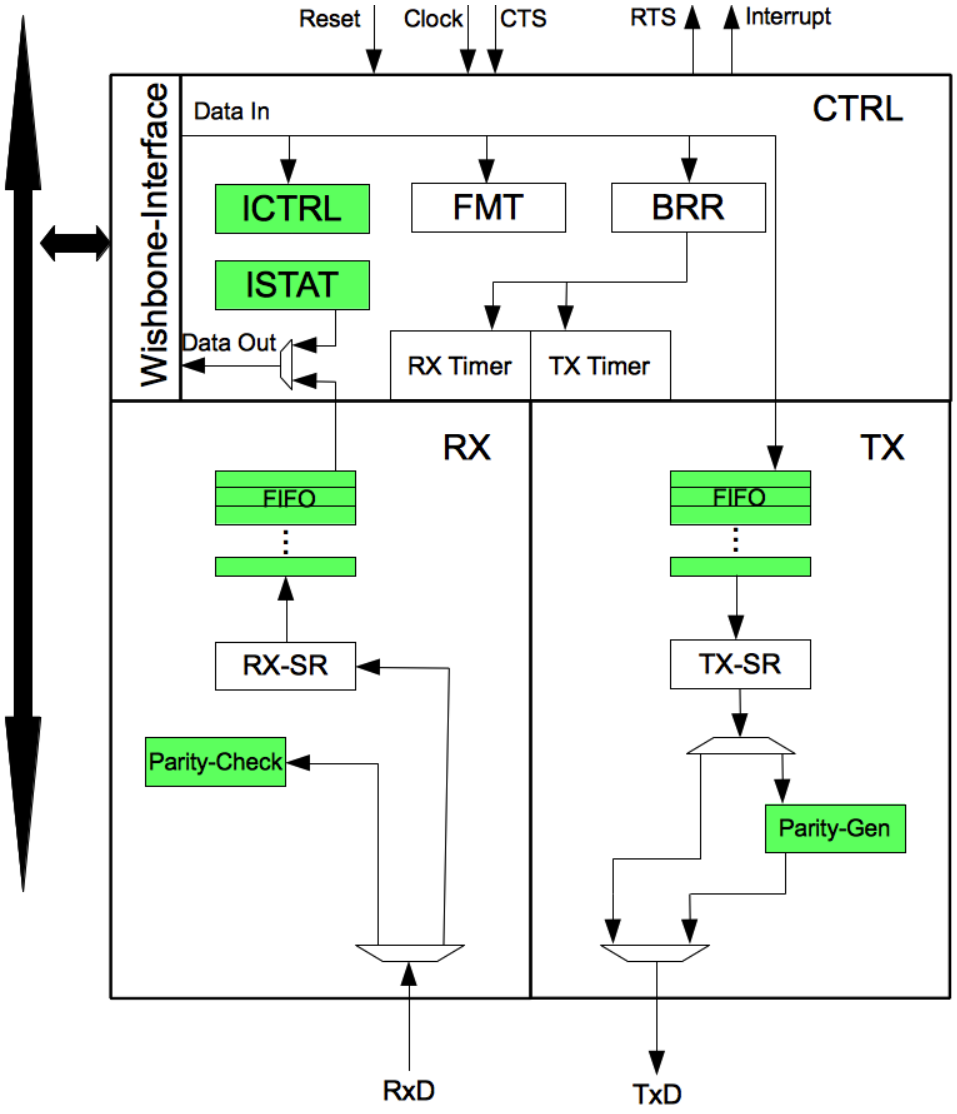
Bereich/Zeitpunkt	Modellierung	Laufzeit
Struktur	Vorhandensein von: - RX/TX - FIFO-Speicher FIFO-Größe	RX-FIFO Trigger-Level
Kommunikation	Paritätskomponenten	Paritätsmodus Paritätsbit aktivieren Framegröße Anzahl Stoppbits Baudrate Flusskontrolle
Interrupts	Vorhandensein von Interrupt-Komponenten (Register und Logik)	Aktivierung der Interrupts

Übersicht

- Einführung
  - Kontext der Arbeit
  - Grundlagen
  - Ziele
- Entwurf des UART
- Implementation
  - VHDL-Komponenten
  - Codegenerierung
- Evaluation
  - UART
  - Codegenerierung
- Fazit

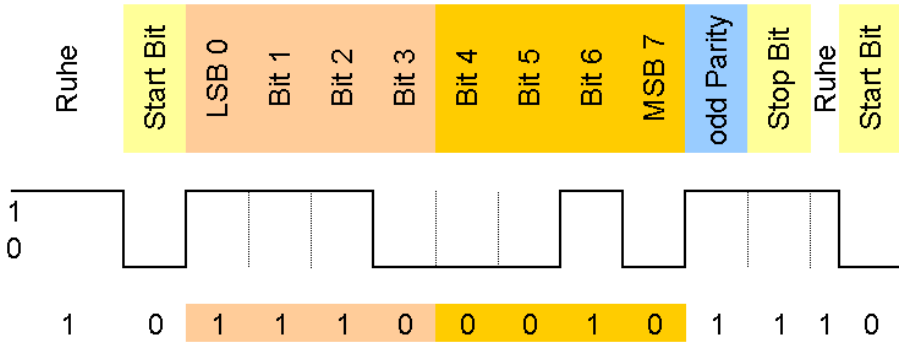
Implementation des UART

- Blockdiagramm des UART:
  - grüne Komponenten optional
  - Control, RX und TX als Hauptmodule



Implementation des UART

- RX- und TX-Einheiten
  - Zustandsmaschinen mit Bitpositionen als Zustandsmenge
  - relativ analog aufgebaut



- Datenpufferung
  - Ohne FIFO nur je ein Datenwort im Schieberegister
  - Mit FIFO zwischen 1 und 64, je nach Konfiguration

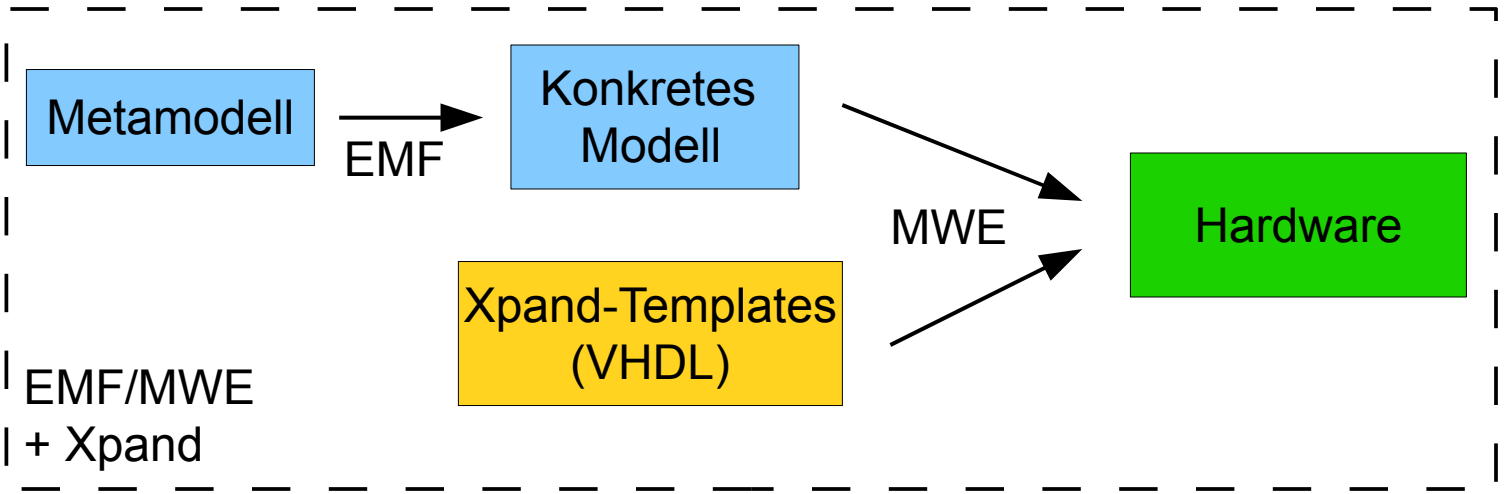
- Steuereinheit
  - Taktteiler für RX und TX
  - Interrupt- und Formatkonfiguration
  - erzeugt Interrupts in Abhängigkeit von RX/TX-Status

Übersicht

- Einführung
  - Kontext der Arbeit
  - Grundlagen
  - Ziele
- Entwurf des UART
- Implementation
  - VHDL-Komponenten
  - Codegenerierung
- Evaluation
  - UART
  - Codegenerierung
- Fazit

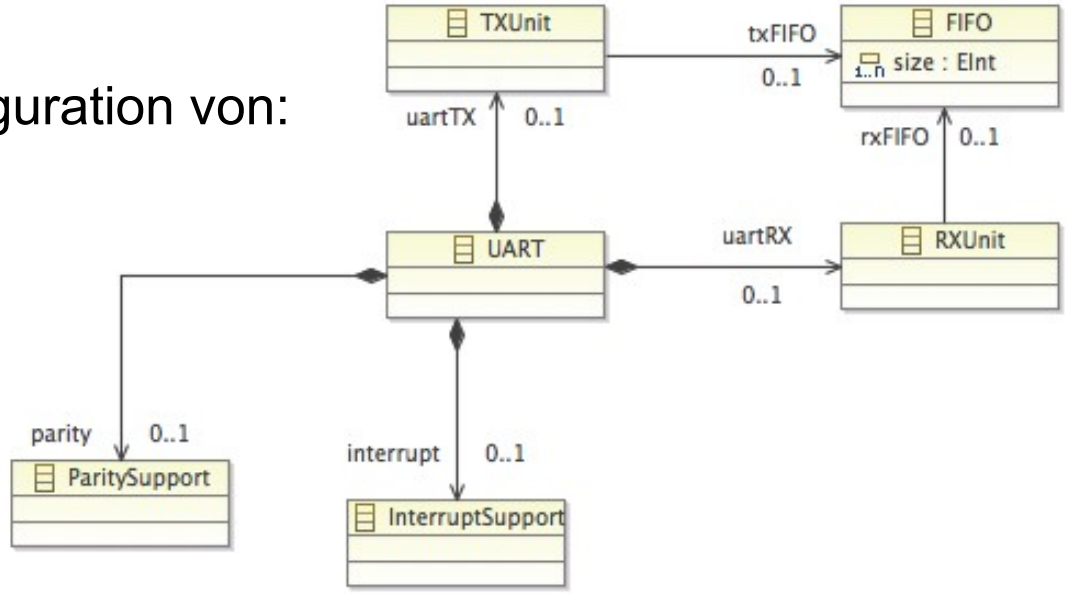
Codegenerierung

- Arbeitsschritte des Workflows
  - Erstellen eines konkreten UART-Modells
  - Validierung des Modells
  - Parser mit Xpand-Templates und Modell instanziieren



Codegenerierung

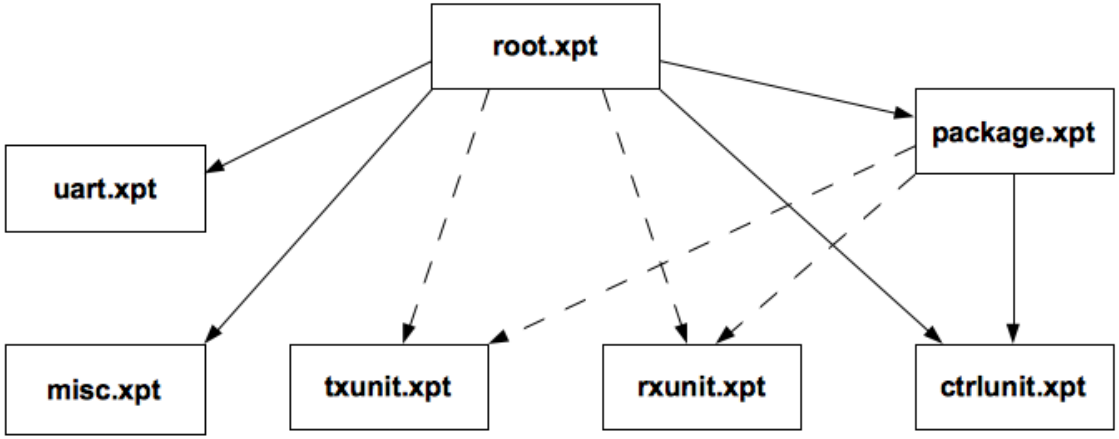
- UART-Metamodell



  - Erlaubt Konfiguration von:
    - Interrupts
    - Paritäten
    - RX/TX
    - FIFOs
  - Grundlage für konkrete Modelle
  - Deckt einige Fehlerfälle ab (Kompositionsstruktur)
  - restliche Validierung über Modellconstraints (oAW Check)

Codegenerierung

- Xpand-Templates
  - in Steueranweisungen gekapselte VHDL-Codesegmente
  - für jedes variable, komplexe Modul des UART ein Template
  - statische Komponenten in einem Template zusammengefasst

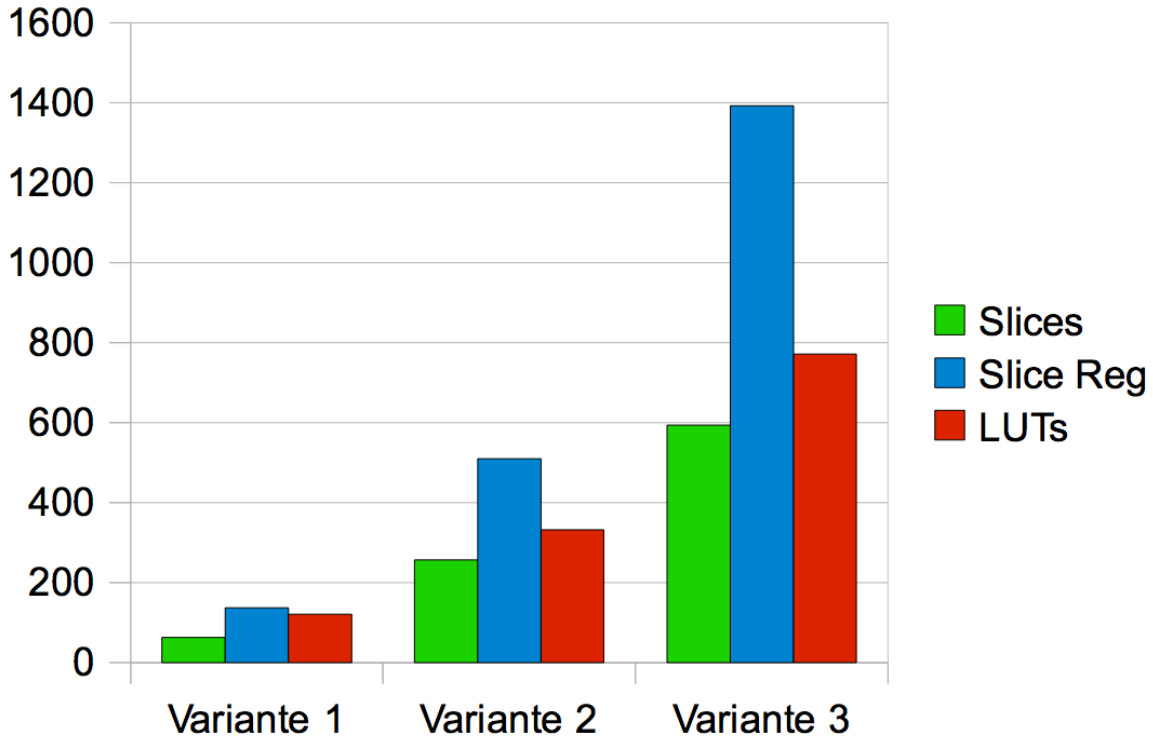


Übersicht

- Einführung
  - Kontext der Arbeit
  - Grundlagen
  - Ziele
- Entwurf des UART
- Implementation
  - VHDL-Komponenten
  - Codegenerierung
- Evaluation
  - UART
  - Codegenerierung
- Fazit

Evaluation des UART

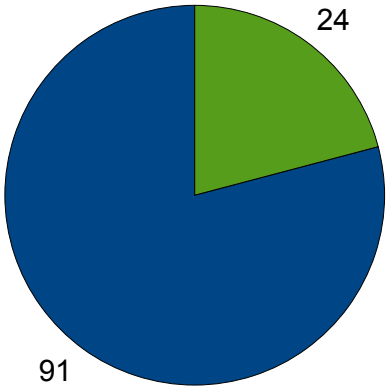
- Ressourcenausnutzung
  - Testvarianten
    - nur TX, kein FIFO (Minimalkonfiguration)
    - RX/TX mit 16 Slot FIFOs + Interrupts
    - RX/TX mit 64 Slot FIFOs + Interrupts + Paritäten



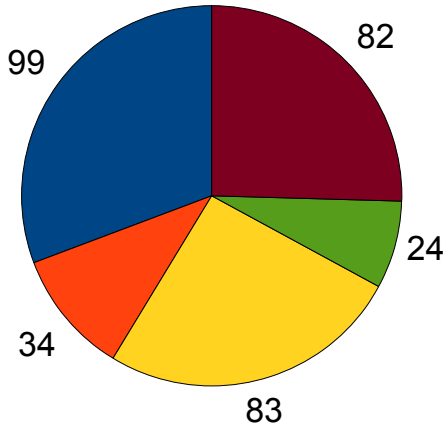
Evaluation des UART

- Detailbetrachtung LUTs

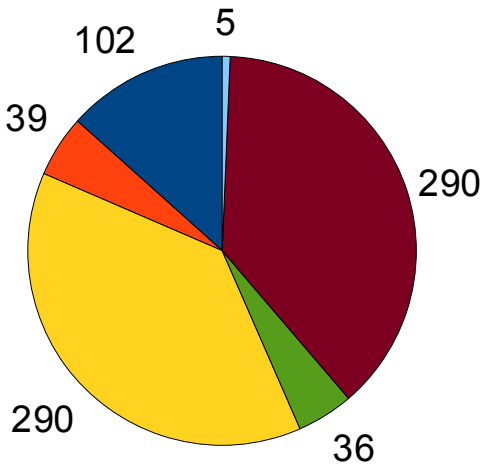
Variante 1

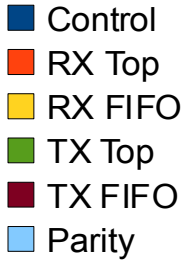


Variante 2



Variante 3





- FIFOs nehmen einen Großteil der Ressourcen ein
- Paritäten sind fast „umsonst“
- restliche Module skalieren mit Funktionsumfang

Evaluation der Codegenerierung

- Kritikpunkte von XVCL
  - Lesbarkeit der x-frames
  - große Zahl lokaler Variablen
  - schwache Typisierung von Variablen (XML)
  - mangelnde Ausdrucksstärke

- Charakteristika von Xpand
  - grundsätzliche Lesbarkeit nicht wesentlich besser 
  - aber: näher am Modell, dadurch weniger Variablen 
  - Variablen sind streng typisiert (Java) 
  - hohe Ausdrucksstärke durch Extensions und Expressions 

Übersicht

- Einführung
  - Kontext der Arbeit
  - Grundlagen
- Entwurf des UART
- Implementation
  - VHDL-Komponenten
  - Codegenerierung
- Evaluation
  - UART
  - Codegenerierung
- Fazit

Fazit

- Entwicklung einer konfigurierbaren UART-Familie
  - Modulare VHDL-Komponenten
  - Konfiguration zur Laufzeit
- Ressourcenverbrauch
  - skaliert gut mit Funktionsumfang
  - allerdings großteils über FIFO-Speicher
- Codegenerierung ohne XVCL
  - einheitlicherer, kompakterer Workflow
  - Xpand insgesamt komfortabler als XVCL
  - Werkzeuge greifen nahtlos eineinander

Vielen Dank für die Aufmerksamkeit!

Gibt es noch Fragen?