technische universität
dortmund

Maßschneiderung von MPSoCs durch Code-Analyse

Abschlussvortrag

Matthias Steinkamp

Vorname.Nachname@tu-dortmund.de

Ziele der Arbeit

- Ableitung der Hardwarekonfiguration aus der Software
  - Höhere Abstraktion der Hardwareentwicklung
- Durchgängige Konfigurierung von Software und Hardware
  - Möglichst optimale Plattform für gegebene Anwendung
- Umfangreiche Werkzeugkette
- Nicht enthalten: Aufteilen der Anwendung auf Cores

Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

HW in SW – Anforderungen

- Mehrere Anforderungen an die Repräsentation
  - Statisch, zur Übersetzungszeit, auswertbar
  - Unterschiedliche Variationen gleicher Bausteine
  - Elementare Schnittstelle für Interaktion mit der Hardware
  - Aufteilung des Adressraums
  - Unterstützung von Unterbrechungsnummern
  - Flexible Struktur für alternative Zielarchitekturen

⟶ C++ Templates!

HW in SW – Lösungsansätze

1. Methodenaufrufe suchen:
   - z.B. in *main(): UART<57600>::write<0>('X');*
   - Instanznummer bei Zugriffen mitliefern → Fehleranfällig
   - Aufrufgraph notwendig

2. Typedefs suchen:
   - *typedef UART<57600> myUART;*
   - Kein Aufrufgraph benötigt
   - Untypisch, dass hierdurch etwas „physikalisches“ instanziiert wird

3. Instanziierung durch globale Objekte:
   - *UART<57600> myUART;*
   - *class Console : public UART<9600> {};*
   - Semantisch vertretbare Lösung; gleichzeitig flexibel

Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

Die Hardware API – Beispiel

```
namespace hw {
namespace api {

template<int Baud>
class UART : public AbstractDevice {
 enum { _Base = 0x101000, _Size = 4096 };

 template<int NR>
 void instance() {
  setAddress(CPU::_IO_Base + _Base + NR * _Size);
 }
};
}}
```

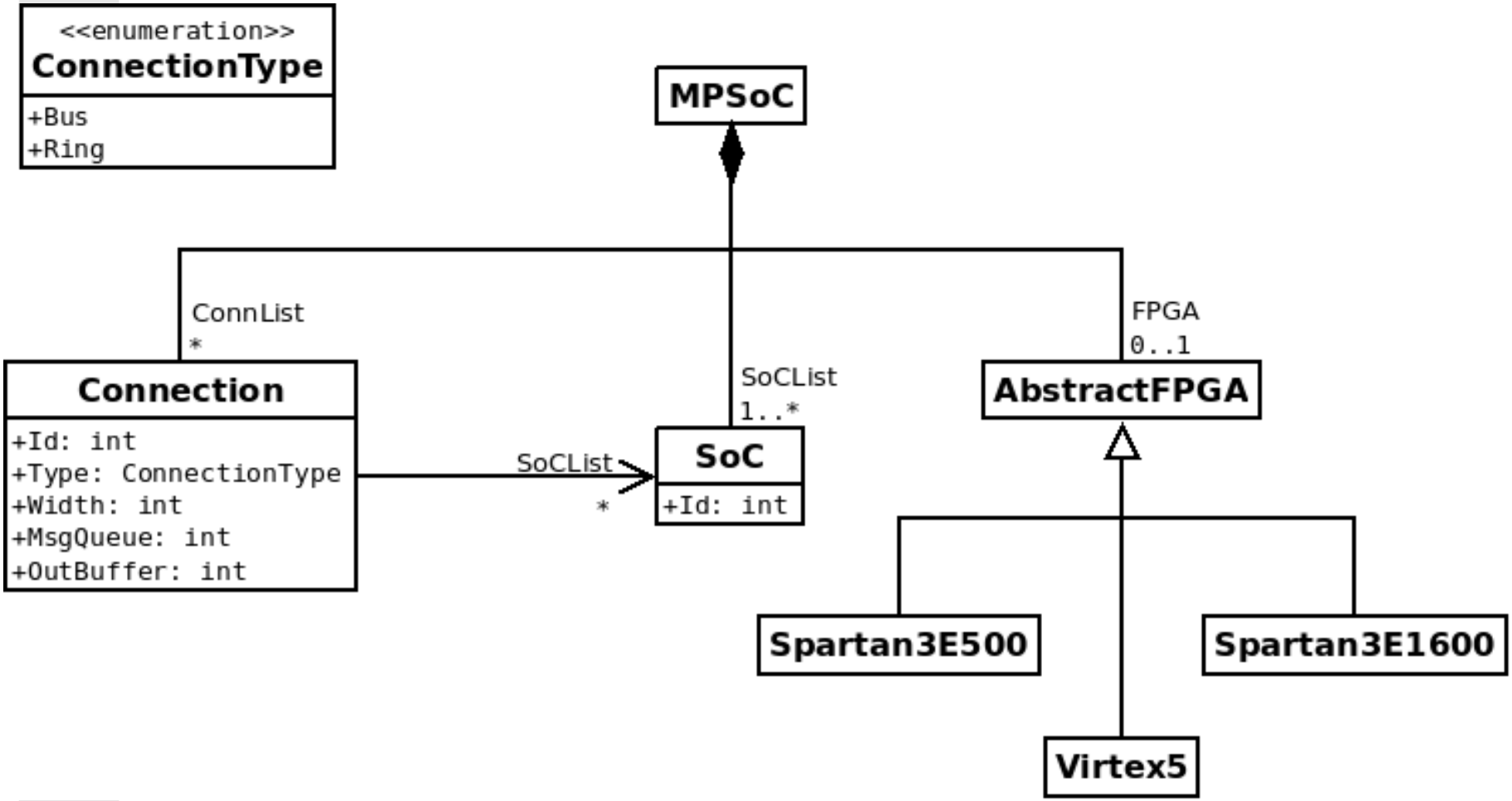
Die Hardware API – Benutzung

```
hw::api::UART<9600> twoUARTs[2];

class Sub : public hw::api::UART<9600> {};
struct Komp {
  Sub s;
} k;


void init() {
  twoUARTs[0].instance<0>();
  twoUARTs[1].instance<1>();
  k.s.instance<2>();
}
```

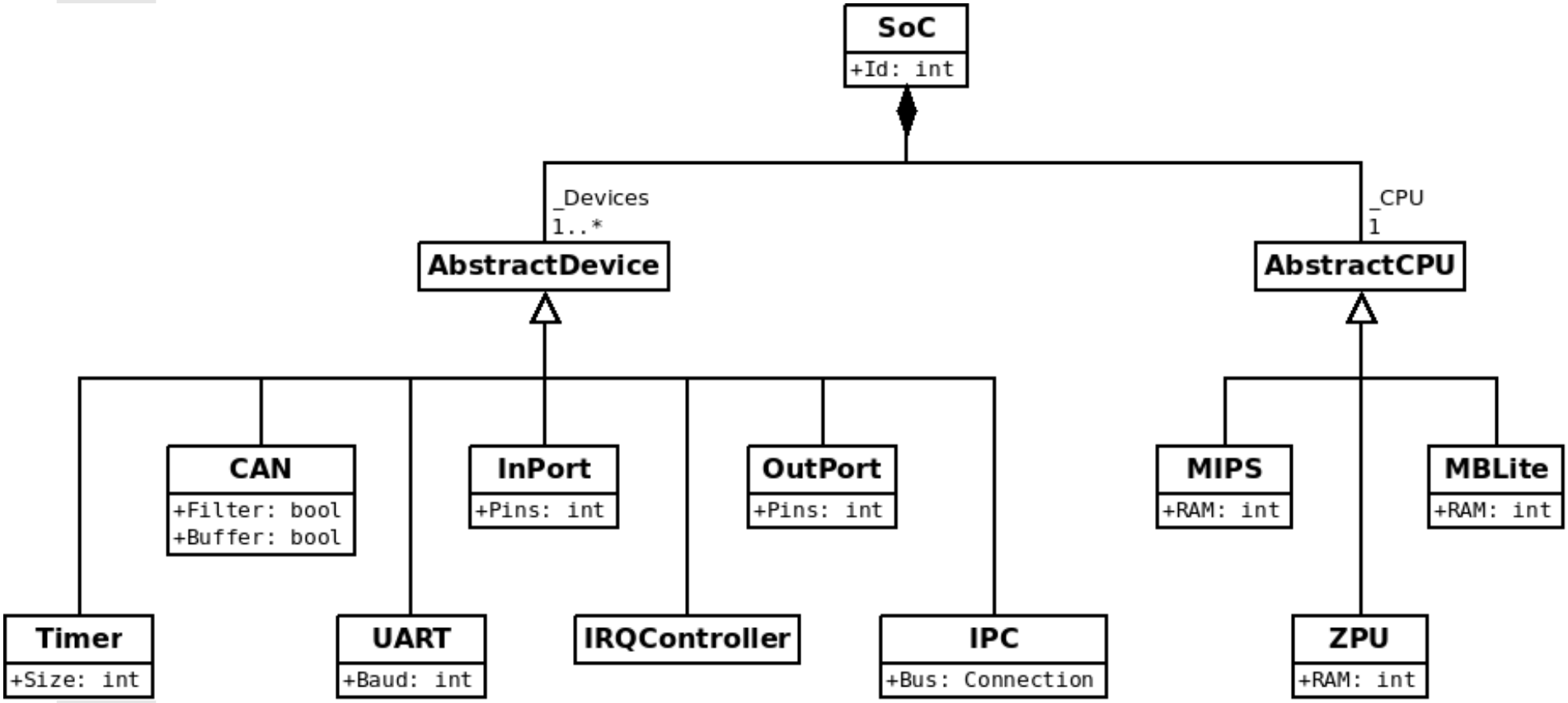
Die Hardware API – MDSD (1)

- Template-Klassen werden aus Metamodell generiert
- Annotation von Register-Offsets möglich
- Statische Fehlerüberprüfung verhindert
  - Falsche Namen
  - Doppelte Einträge
  - Fehlende Angaben (z.B. Kapazität eines FPGAs)

Die Hardware API – MDSD (2)



Die Hardware API – MDSD (3)



Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

Parser

- Basiert auf PUMA
- Als Eingabe dienen N Verzeichnisbäume
  - Pro Baum wird ein SoC erstellt
  - Alle gewobenen Quelltextdateien werden bearbeitet
- „parser.config“ konfiguriert die Suche
  - Enthält PUMAs Projektpfade (Optionen „-d“ und „-p“)
  - Enthält individuelle Include-Anweisungen

Parser – Suche

- Sucht Vorkommen der Hardware API
  - Iteration über alle CAttributeInfo und CFunctionInfo
  - Untersucht die Klassen der Objekte
  - Beliebige Kombination von Komposition und Vererbung
  - Vermerkt geforderte Unterbrechung
- Sucht die Initialisierung der Hardware API
  - Über den Syntaxbaum (CVisitor)
  - Sichert alle Initialisierungen zu einer HW-API-Instanz

Parser – Konfiguration

- Falsche Initialisierung kann behoben werden
  - Dialog per Konsole mit Entwickler
  - Manipulation des Syntaxbaums
- Generiert Unterbrechungsnummern:

```
template<>
class IRQ<UART> {
  enum { NUM = 0 };
};

template<>
class IRQ<Timer> {
  enum { NUM = 1 };
};
```

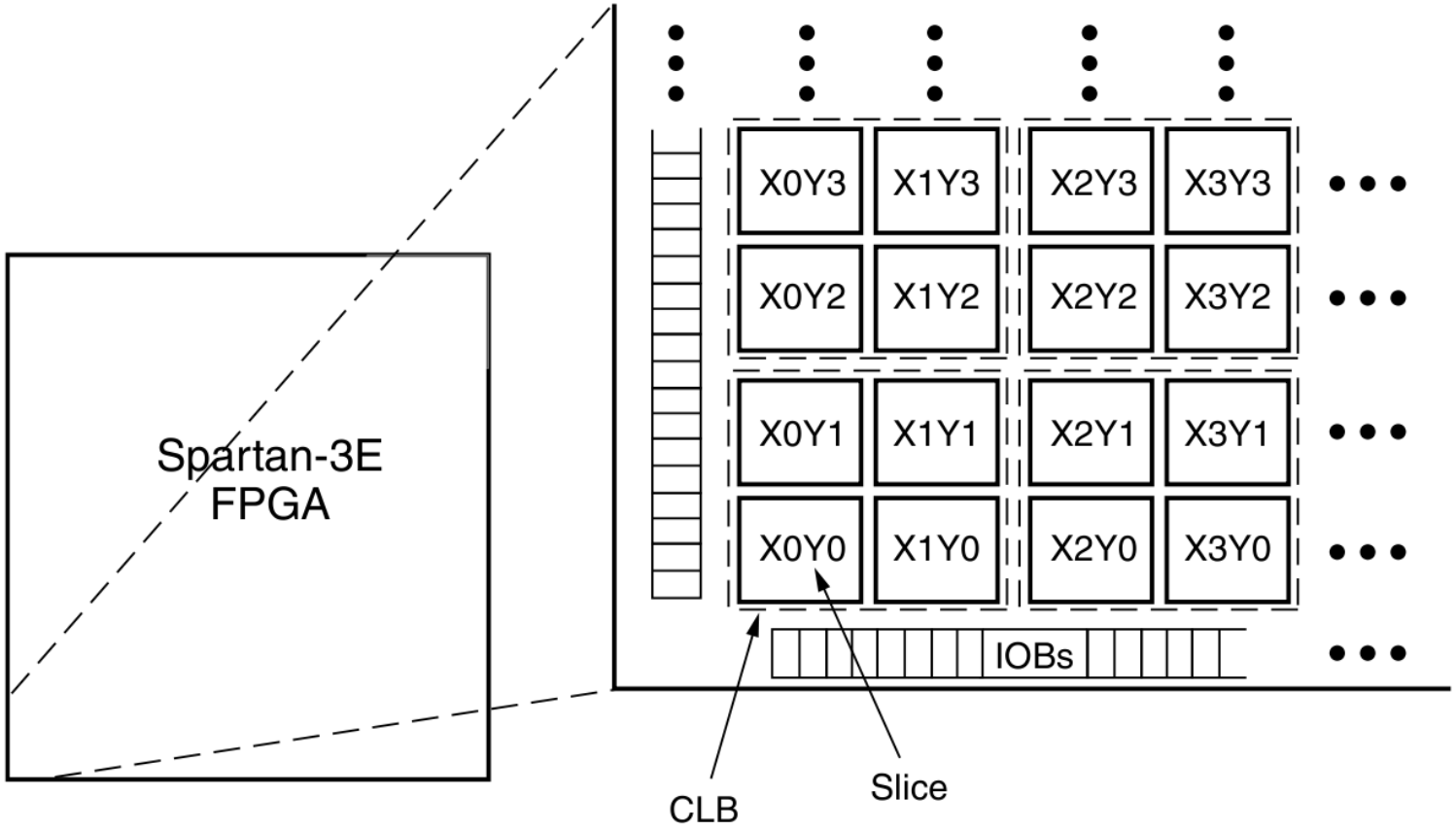
Datei IRQ_generated.h

Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

Ressourcenbedarf – Definition

- Elementare Strukturen der FPGAs können Kosten sein
  - Eindeutige Zuordnung zur High-Level-Komponente
  - Kosten := Verbrauch von Block RAM und LUTs



Quelle: Xilinx Spartan-3E FPGA Family Datasheet

Ressourcenbedarf – Implementierung

- Kosten im Metamodell annotieren
  - Namenskonvention: Kosten beginnen mit „_C_“
  - Mögliche Kosten: LUT4, BRAM2 (Spartan), LUT6, BRAM4 (Virtex5)
- Vermessung jeder einzelnen Komponente
  - Alle Variablen beachten
  - Redundante Messungen für Spartan3 und Virtex5
  - Lineares Modell
- Berechnung auf Basis des MPSoC-Modells in oAW
  - Modell ist ein Baum
  - (Fast) jeder Knoten definiert Kosten
  - Traversierung liefert Gesamtkosten

Ressourcenbedarf – Beispiele

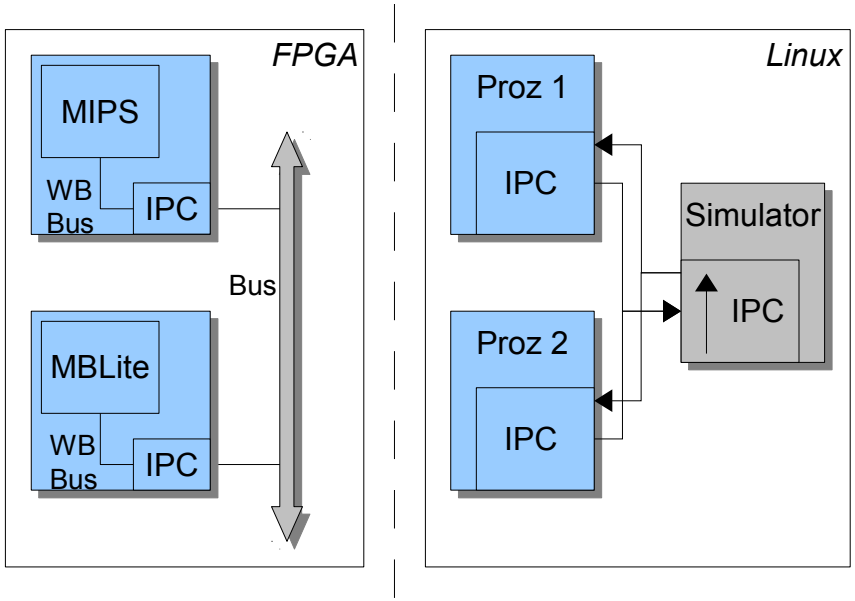
```
Integer getCosts(UART uart, SoC _soc, String type) :
  switch (type) {
    case "_C_LUT4" : (uart.Baud <= 38400 ? 89 : 75)
    case "_C_LUT6" : (uart.Baud < 38400 ? 65 : 62)
    default : 0
  };

Integer getCapacity(Spartan3E1600 fpga, String type) :
  switch (type) {
    case "_C_LUT4" : (29504)
    case "_C_BRAM2" : (36)
    default : 0
  };
```

Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

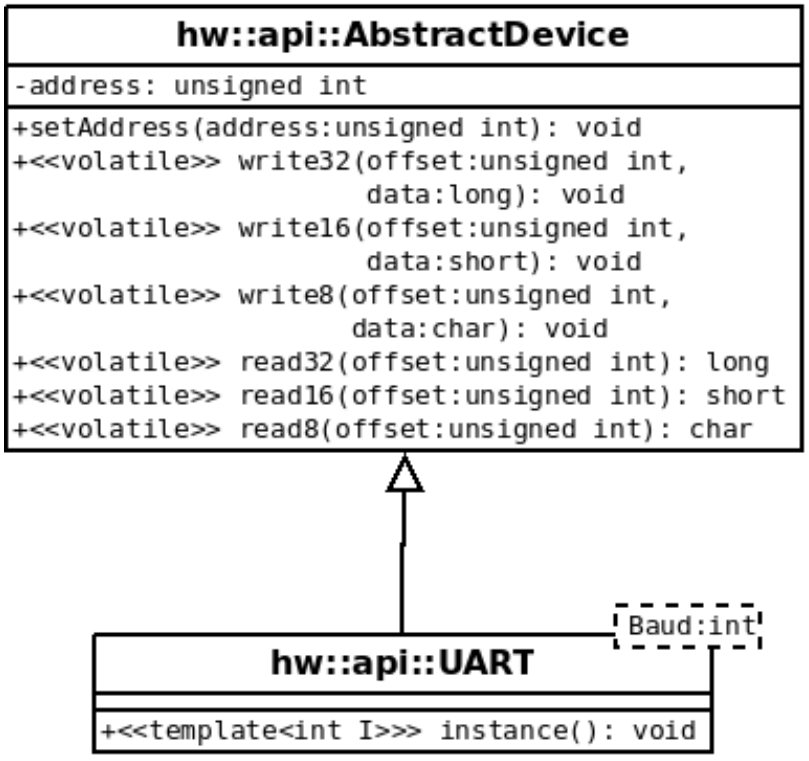
Simulator

- Simuliert das MPSoC mit Linux
  - Quelltext für x86 kompilieren!
  - Integriert in Björns CiAO Gastsystem
- Systemaufrufe bilden Hardware nach
  - IPC durch Message Queues
  - UART durch FIFOs
- Hauptprozess koordiniert:
  - Liest MPSoC-Modell ein
  - Startet SoC-Prozesse
  - Stellt IPC-Kommunikation her

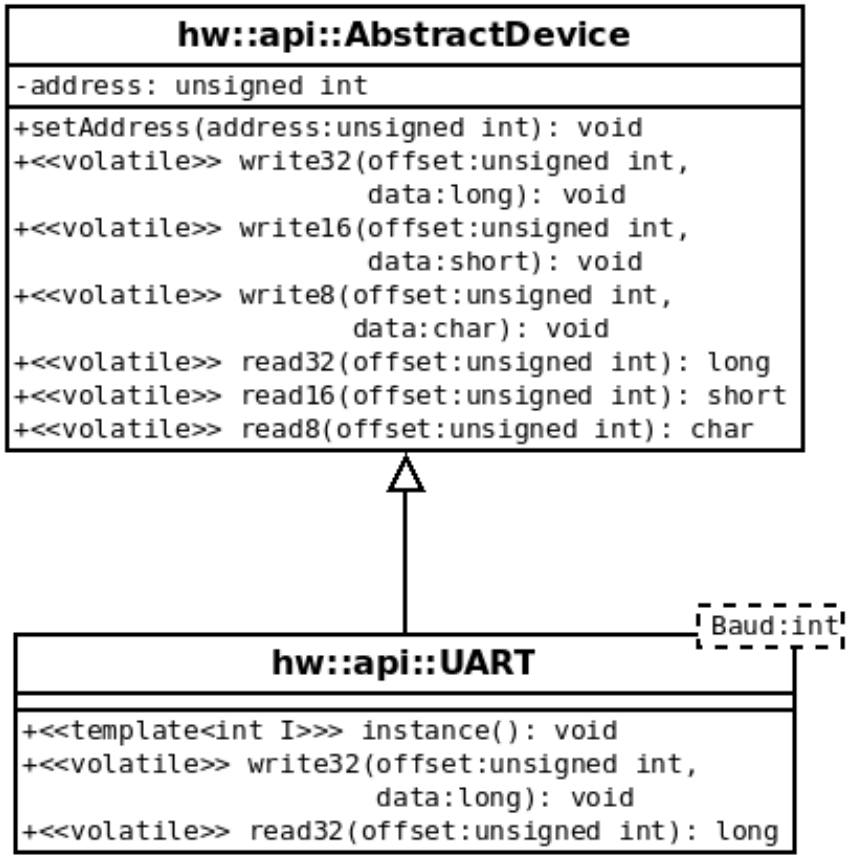


Simulator – Alternative HW API

Implementierung durch Hardware



Alternative Implementierung



Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

Werkzeugkette

- Realisiert in „lava.sh“
- Parameter:
  - -src=<Verzeichnis> Verzeichnis der verteilten Anwendung
  - -nc Den Quelltext nicht kompilieren (CiAO)
  - -list Listet unterstütze FPGAs auf
  - Letzter Parameter Name des FPGAs oder „sim“
- Integriert alle Tools:
  1. Weben und Parsen der verteilten Anwendung
  2. Generierung des SPCs mit oAW
  3. Kostenberechnung und Modellvalidierung
  4. Kompilieren des manipulierten Quelltextes
  5. Simulation oder Xilinx starten

Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

Evaluation – Durchgängige Konf'keit

- Automatische Aufteilung der Anwendung nicht realisiert
- Parser benötigt mehrere Verzeichnisbäume
- Notwendige manuelle Arbeiten:
  - Anzahl und Prozessortyp der SoCs bestimmen
  - Teilen der Anwendung in separate Teilprojekte
  - Anpassen der config.xml
  - Neue Konfigurierung des Betriebssystems (u.A. Treiber auswählen)

Evaluation – Ressourcenbedarf (1)

- Vergleich von Messung und Schätzung
- Folgende Testsysteme:

System	Prozessoren	Peripherie
1	2 MBLite Je 8KB RAM	1 UART, 57600 Baud 1 CAN, mit Filter 2 IPC Controller, 32 Bit
2	4 MBLite Je 8KB RAM	4 CAN, mit Filter 4 IPC Controller, 32 Bit
3	15 ZPU Je 2KB RAM	9 UART, 57600 Baud 21 IPC Controller, 32 Bit

Evaluation – Ressourcenbedarf (2)

- BRAM:
  - passt für Spartan3 immer, bei Virtex5 optimiert Xilinx
- LUTs:

System	Spartan3			Virtex5		
	Sch.	Mess.	Diff.	Sch.	Mess.	Diff.
1	7843	7793	+0,69%	5687	5615	+1,28%
2	17496	17577	-0,46%	12852	12897	-0,35%
3	23886	24755	-3,51%	17745	19508	-9,04%

Evaluation – Simulator

- Alternative Plattform zum FPGA
- Vorteile:
  - Weniger Fehlerquellen
  - Zeitersparnis dank fehlender Synthese (15 Minuten bis X Stunden)
  - Besseres Debugging
  - Keine Ressourcenbeschränkung
- Nachteile:
  - Keine echte Parallelität
  - Abhängigkeit vom Hostsystem und dessen Auslastung / Scheduling
  - Keine Echtzeitfähigkeit

Inhalt

- Repräsentation von Hardware in Software
- Hardware API
- Parser
- Abschätzung des Ressourcenbedarfs
- Simulator
- Werkzeugkette
- Evaluation
- Fazit

Fazit

- Durchgängige Konfigurierung von HW und SW möglich
  - Wenn auch (noch) nicht mit CiAO
- Werkzeugkette ermöglicht Entwicklung von MPSoCs auf hoher Abstraktionsebene
- MDSD erlaubt flexible Änderungen und Erweiterungen
- Ausblick:
  - Es fehlt an richtigen Anwendungen!
  - Die (optimale) Aufteilung der Cores muss integriert werden

Fragen?