Architektur einer Flexiblen, Wiederverwendbaren Testbench zur Verifikation Paketverarbeitender Hardware in SystemC

Stephan Kubisch, Harald Widiger, Ronald Hecht,
Dirk Timmermann, Martin Siemroth
Institut für Angewandte Mikroelektronik und Datentechnik
Universität Rostock

10. GI/ITG/GMM MBMV 2007
5.-7. März 2007, Erlangen, Deutschland

Inhalt

1. Motivation

Simulation & Verifikation

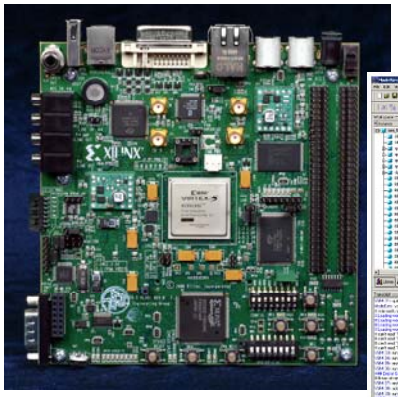
- Designs werden komplexer!
- Verifikation ist umso aufwendiger bzgl. Zeit und Kosten

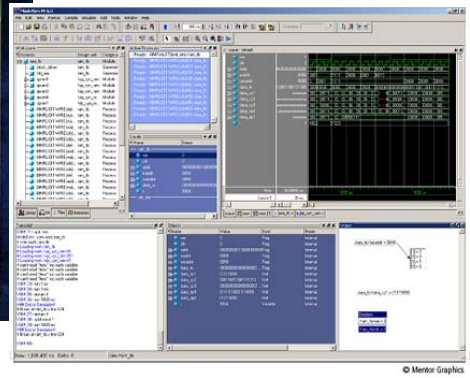
→ **Ziel 1: Wiederverwendbarkeit**

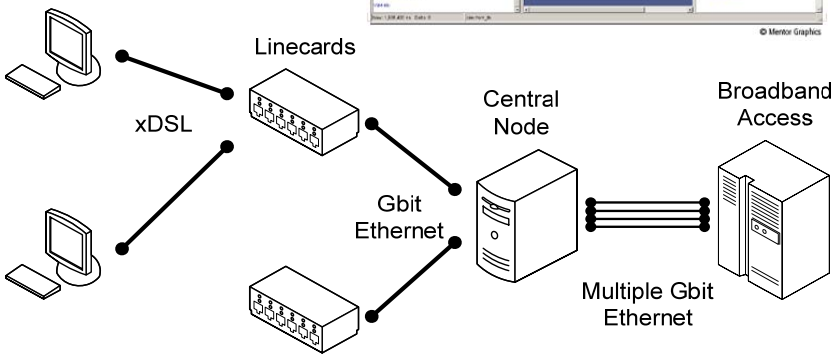
- für versch. Designs
- für versch. Schritte im Design Flow

→ **Ziel 2: Handhabbarkeit**

- Abstraktion, Objektorientierung



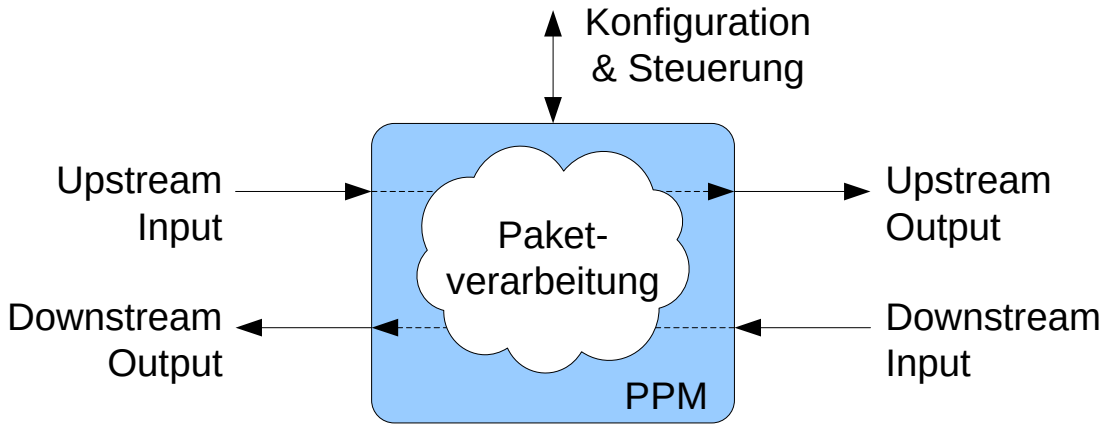




Eigene Forschungsarbeit

- Kooperation mit Siemens Networks (2Q/07→ Nokia Siemens Networks)
- Mechanismen & Baugruppen speziell in Zugangsnetzwerken
- ähnliche, wiederkehrende Szenarien und Testfälle

2. Hintergrundinformationen

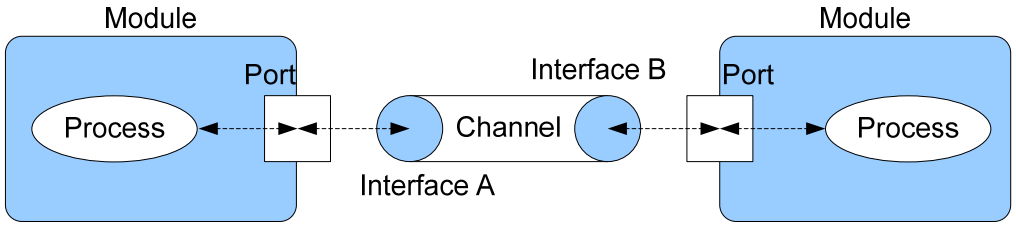


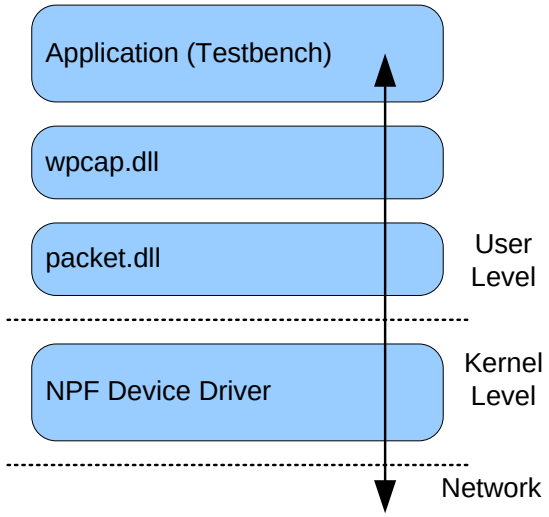
Paketverarbeitende Systeme als Device-Under-Test (DUT)

- Ähnliche Interfaces
  - Datenpfade “von links nach rechts”, Konfig.- & Steuerleitungen
- Gleiche/ähnliche Basis-Protokolle
- Schichtenhierarchie nach OSI
- Strukturierte, gekapselte Informationseinheiten nach OSI

→ **Gemeinsamkeiten nutzbar machen & wiederverwenden**

2. Hintergrundinformationen





SystemC

- C++, IEEE Standard
- Beschreibung von HW-Strukturen
- Basiskonzept
  - Module, Prozesse
  - Schnittstellen, Ports, Kanäle
- SCV, TLM und MS Bibliotheken als zusätzliche Erweiterungen

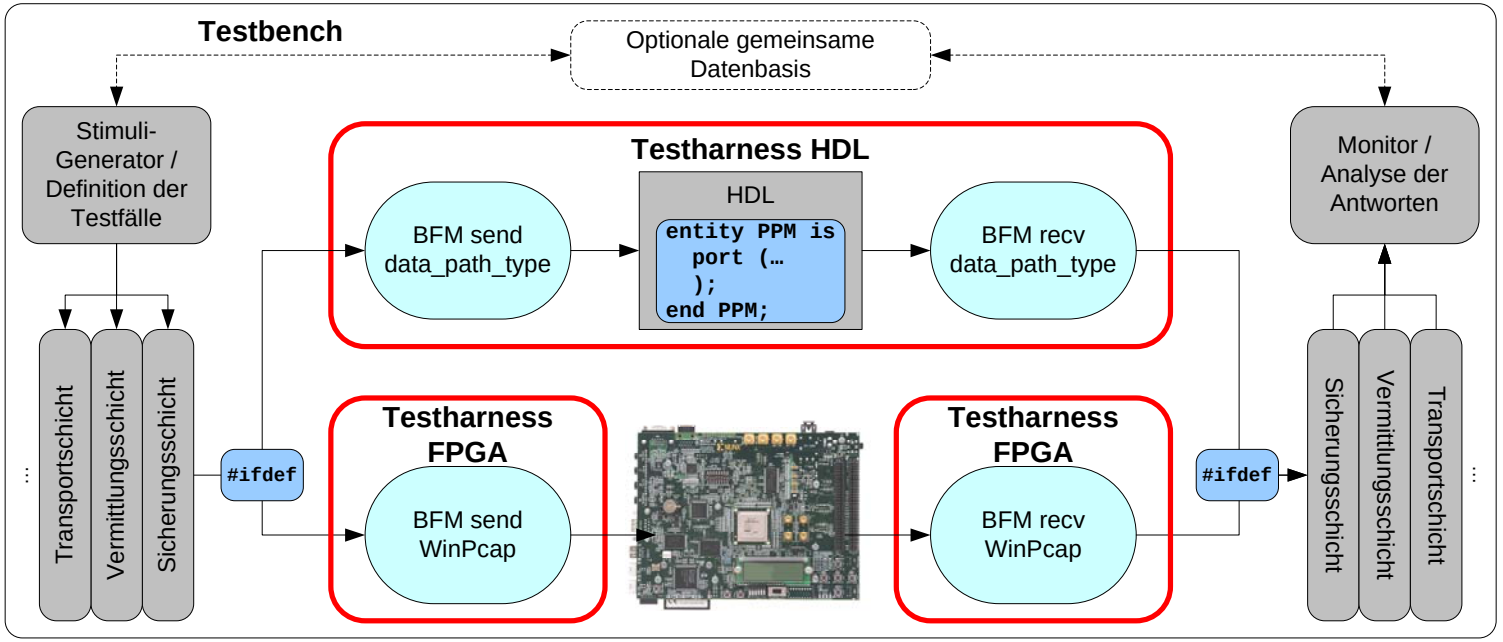
WinPcap

- C++ Bibliothek
- Zugriff auf physikalische Netzwerkschnittstelle
- bekannt durch libpcap (UNIX)
- z.B. Snort & Wireshark

3. Aufbau der Testumgebung

Gesamtstruktur

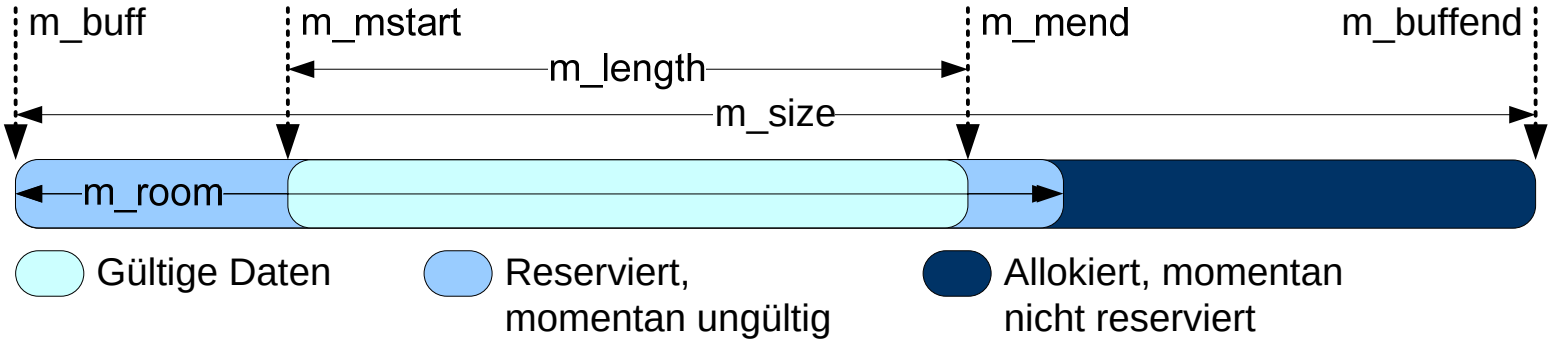
- Zweiteilung der Testumgebung
  - Testbench (= Datenebene)
    - Stimuli-Erzeugung und Auswertung der Antworten
  - Testharness (= Zeitebene)
    - Stimuli-Übertragung, abhängig vom Interface
- Compiler-Direktive zur Unterscheidung zwischen Software und FPGA



3. Aufbau der Testumgebung

Message-Klasse

- Container für Protokoll-Informationen
- Realisierung der typ. Struktur von Informationseinheiten nach OSI
- Methoden zur Manipulation eines Message-Objektes (push(), pop(), pull(), trim(), length(), …)
- Struktur
  - allokierter Speicherbereich des Objektes
  - für Nutzdaten reservierter Speicherbereich
  - aktuell gültige Daten im reservierten Bereich

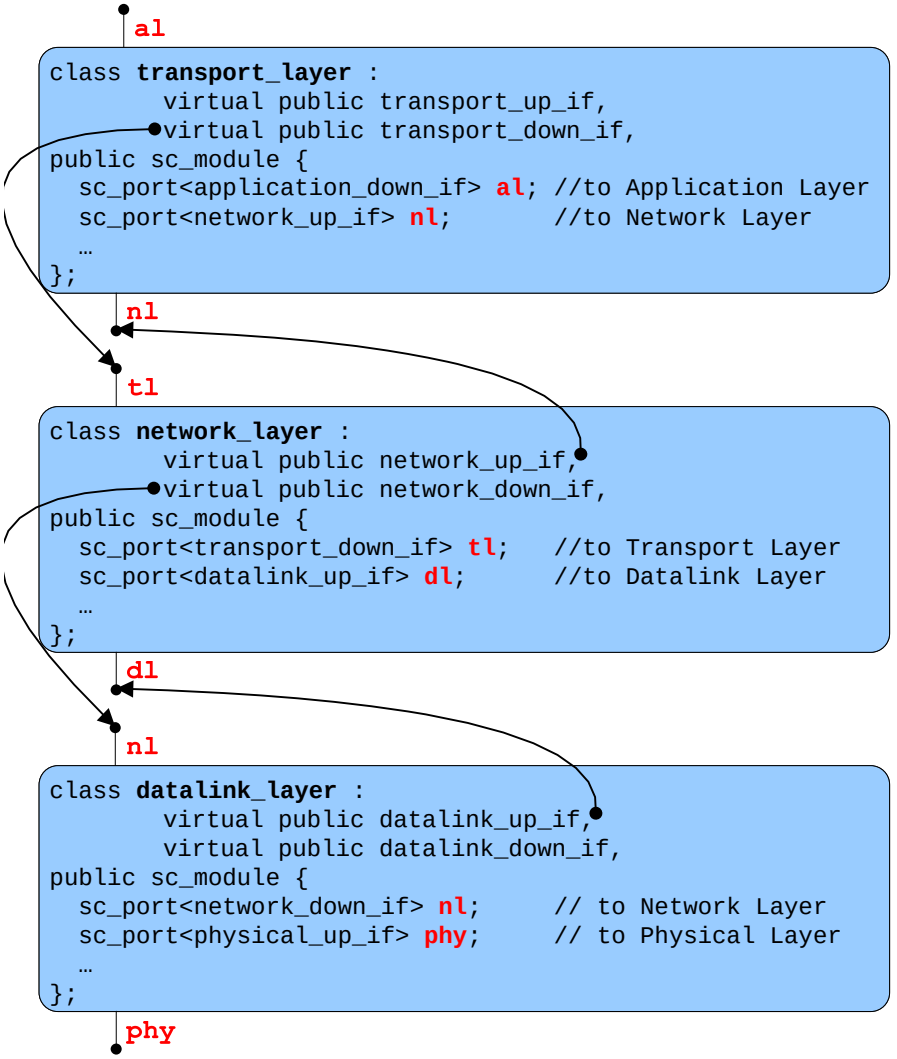


3. Aufbau der Testumgebung

Umsetzung der Protokoll-Hierarchie

- Für Stimuli-Erzeugung und Auswertung
  - Ver-/Entpacken von Paketen
  - Container für Protokolle
- Eine <class> pro OSI Schicht
  - Implementierung vererbter Schnittstellen
  - Ports zu über- & untergeordneter Schicht
  - Kommunikation über Ports & Schnittstellen
- ☝ nicht zwingend OSI

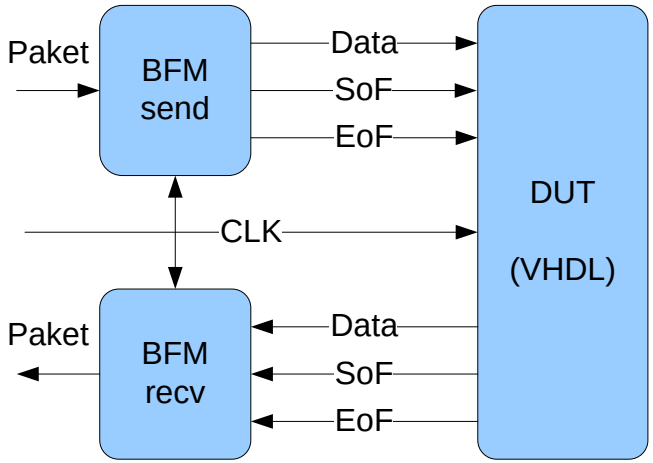
→ beliebige Schichten/Protokolle

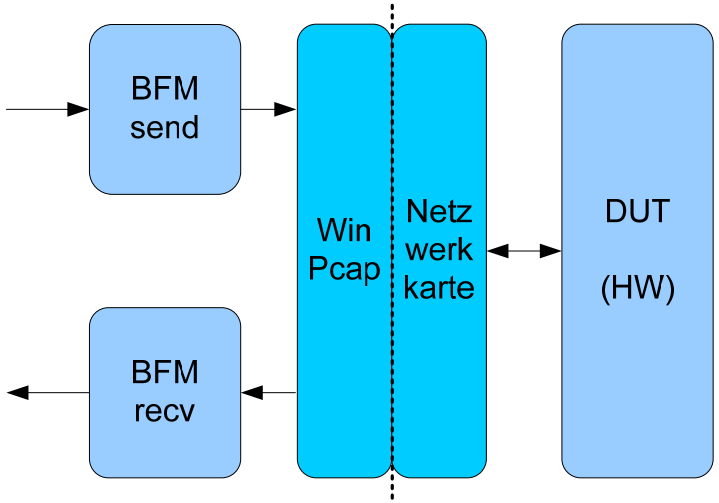


3. Aufbau der Testumgebung

Bus Functional Models

- Übergang von Daten- in Zeitebene
- BFM implementiert zwei Schnittstellen
  - Zur Testbench, Umwandlung einer Message in “echte” Bits/Bytes
  - Protokoll der Schnittstelle des BFMs → Individuell für ein DUT
- Beispiele:
  - BFM für eine VHDL-Entität
  - BFM für WinPcap-Schnittstelle





4. Vor- & Nachteile

Vorteile

- Die vorgestellten Mechanismen sind „nur“ flexible Container
  → Anpassung an/Erweiterung um individuelle Funktionen
- C++ & SystemC sind mächtige Sprachen
  → einfachere Beschreibung komplexer Testfälle
- Nutzung von WinPcap
  → Verifikation von PPMs auch auf FPGA Prototypen (gleiche Testfälle)

Nachteile

- Verschiedene C++ Compiler sind nicht zwingend kompatibel,
  z.B. ModelSim’s gcc oder VisualC++
  → Kompromisse bzw. Umwege im Code
- SystemC ist nur begrenzt synthesefähig
  → Entwickler müssen nach wie vor 2 Sprachen beherrschen

4. Zusammenfassung

- Gerüst einer flexiblen Testumgebung
- Nachbildung
  - Schichten-Hierarchie
  - typische Struktur von Informationseinheiten
- speziell für paketverarbeitende Systeme
  - Internet Paket Processing
  - Network-on-Chip basierte Systeme
  - Datenstrom-basierte Systeme (Grafik, Verschlüsselung,…)
- Verifikation von HDL-Modellen in Software
- Prototyp-Verifikation z.B. in FPGAs auf Entwicklungsboards

Vielen Dank!

Fragen?

Backup

m_buff = Pointer auf Start des allokierten Speicherbereiches (fix)
m_buffend = Pointer Ende des allokierten Speicherbereiches (fix)
m_mstart = Pointer auf Start der aktuelle gültigen Daten der Nachricht (SoF, dynamisch)
m_mend = Pointer auf Ende der aktuelle gültigen Daten der Nachricht (EoF, dynamisch)
m_size = gesamter für die Nachricht allokierter Speicher/Puffer
m_len = aktuelle Länge einer Nachricht (headers+payload)
m_room = bei Initialisierung für Nachricht reservierter Speicher/Puffer
m_tail = m_size – m_room, Größe des unreservierten aber allozierten Speichers (eine art reserve)