Audiosignalverarbeitung mit FPGA

Oswald Berthold

Humboldt-Universität zu Berlin
Institut für Informatik
Lehrstuhl Signalverarbeitung und Mustererkennung

2. März 2010

Outline

Audiosignalverarbeitung mit FPGAs

- Projekt im Rahmen des Seminars “Hardware der Signalverarbeitung”, WS09/10, F. Winkler
- Hardware Plattform: Xilinx Spartan und Virtex
- Grundlagen des Entwurfs: Toolchain (EDK, ISE)
- System: Framework, Schnittstellen, vorhandene Komponenten

Aufgabenstellung

- Entwicklung einer Schwellwert-getriggerten Audioübertragung über Netzwerk (z.B. Babyfon)
- Allgemeiner: ein System, dass ein einkommendes Signal analysiert und bei Eintreten bestimmter Ereignisse (z.B. Überschreiten eines Schwellwerts) die Übertragung dieses Signals, auch kombiniert mit anderen Signalen, über ein IP-Netzwerk auslösen kann.

Hardware

- Das System soll auf der Xilinx Virtex II Pro Plattform implementiert werden.
- Das xupv2p-Board stellt neben dem Chip die notwendigen Schnittstellen zur Verfügung
  - Audio-Interface (AC97)
  - Netzwerk Interface (NIC)
- Virtex II Pro verfügt über zwei eingebettete PPC-Prozessoren

Überlegungen zu Architektur und Vorgehen

- Samples müssen vom Audio-Interface abgeholt, in einem hardware-unabhängigen *Kern* verarbeitet und anschliessend über die Netzwerkschnittstelle verschickt werden.
- "komplexe" Schnittstellen werden von Prozessoren (PPC, Microblaze) bedient
- Durchführung in zwei Phasen
  1. Basissystem mit "Null-Kern" (Identität) stellt den vollständigen Signalpfad her
  2. Anschliessende Einbindung eines in massgeschneiderter Hardware realisierten Signalverarbeitungs-Kerns (Strukturskizze auf der folgenden Seite)

System Block-Diagramm

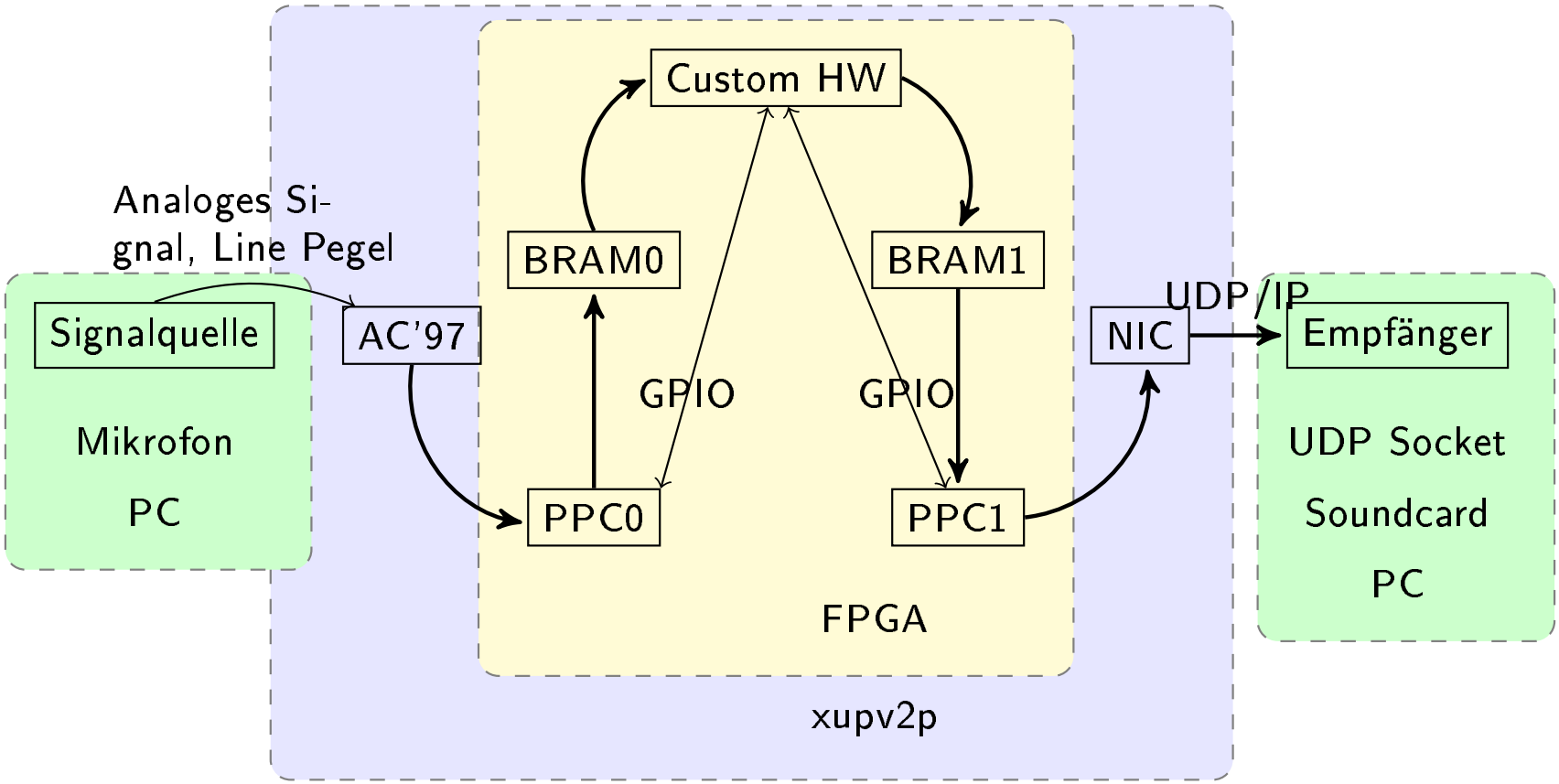

Abbildung: Vorgriff: Struktur nach der Implementierung

Implementierung

Im folgenden werden einige Details der Implementierung erläutert.

- Programmierung der eingebetteten PPC-Prozessoren
- Audio-Schnittstelle
- Netzwerkschnittstelle
- Schnittstelle zwischen beteiligten Komponenten
- BRAM-Kopierer in VHDL
- Schwellwerterkennung in VHDL
- System-Konfiguration über Netzwerk

Toolchain

- xupv2p Board mit Version 10 der Xilinx Tools bespielbar
- EDK: Hardware/Software Codesign
  - Hardware-Spec über system.mhs: enthält Komponenten (IP cores) und deren Verdrahtung
  - Software-Spec über system.mss: Konfiguration der Prozessoren, notwendigen Treiber in Abhängigkeit von der verwendeten Hardware, Software-Bibliotheken, OS, ...
  - Softwareentwicklung für Prozessoren
  - Build-Prozess wird über Makefiles aus EDK gesteuert
- ISE: zur Entwicklung, Simulation und Synthese einzelner Hardware-Komponenten / IP Cores.
- xmd: Xilinx Debugger lädt die Software-Komponenten in den Prozessor

Audio-Schnittstelle

- AC97 Chip auf xupv2p: LM4550VH
- Basis: Verschiedene Xilinx-Beispiele die den AC97 Chip in einem Microblaze-basierten System verwenden
- Weiterverwendbare Komponente: OPB basierte AC97 Komponenten aus audio.zip (http://www.xilinx.com/univ/xupv2p.html)
- Minimales Programm für PPC
  - Lesen der Audiosamples über eine vom Treiber zur Verfügung gestellte FIFO
  - Serielle Kommunikation zum Debugging
  - Schreiben der Samples auf den DAC

Audio-Schnittstelle

- Software Treiber für opb_ac97, FIFO und Puffer basiertes Lesen und Schreiben
- Im FIFO-Modus ist die Ausgabe je ein 16 Bit Sample für den linken und rechten Kanal, die in ein int32 gepackt sind

```
Xuint32 *aucur;
aucur = (Xuint32*) \
   XPAR_PPC405_1_AUDIO_BRAM_CNTLR_BASEADDR+2;
for ( i = blocksize)
   *aucur = XAC97_ReadFifo(XPAR_OPB_AC97_0_BASEADDR);
   XAC97_WriteFifo(XPAR_OPB_AC97_0_BASEADDR, *aucur);
   aucur++;
// irgendwas mit dem block machen
```

Netzwerkschnittstelle

- Komponenten der Ethernet NIC: MAC (im FPGA), PHY (LXT972A)
- Basis: Umfangreiches Xilinx-Demo mit verschiedenen Netzwerkanwendungen: Webserver, Echo-Server, ...
- Konfiguration des xilkernel-Layers (minimales OS) für den PPC Prozessor (Ethernet Treiber)
- LwIP Bibliothek für IP Stack
- UDP

Hardware-unabhängige Schnittstelle

- Voraussetzung um signalverarbeitende Hardware in den Signalpfad zu bekommen, m.a.W. die Samples müssen aus dem Prozessor in den SV-Kern gelangen
- Optionen
  - FSL (Fast Simplex Link), Microblaze only
  - GPIO (Overhead auf Prozessorseite)
  - BRAM
  - Bus (OPB, PLB), Aufwand

BRAM-Schnittstelle

- Double-Buffer Prinzip
  - Füllen des ersten Puffers
  - Signal Puffer 1 voll
  - Lesen des ersten Puffers
  - Signal Puffer 1 gelesen
  - analog mit Puffer 2
- Optionen zur Synchronisation (Zustandsübermittlung)
  - reservierte Adressen im BRAM
  - GPIO Leitungen

BRAM-Schnittstelle Automat

Nochmal als Automat

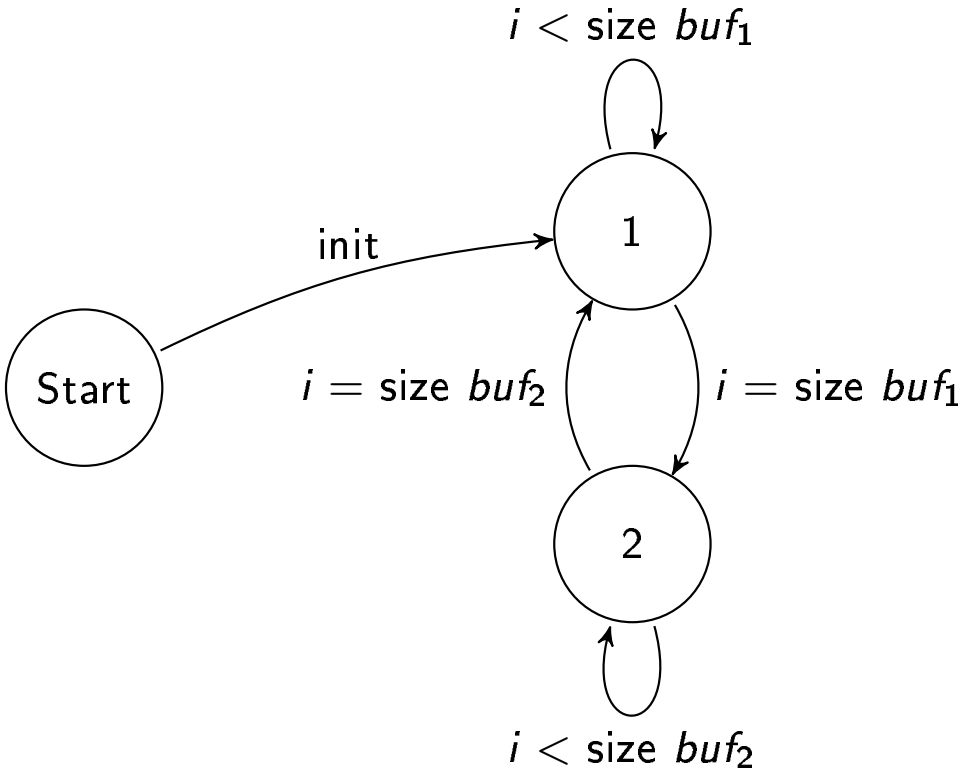

Abbildung: Automat zur 2-Puffersynchronisation

BRAM-Kopierer in VHDL

- einfacher BRAM-Kopierer: lesen des Quell-BRAMs und Schreiben des Ziel-BRAMs
- synchrone Addresserzeugung
- Zustandsübermittlung über GPIO

Schwellwerterkennung in VHDL

- Realisierung eines einfachen Schwellwerttriggers in VHDL

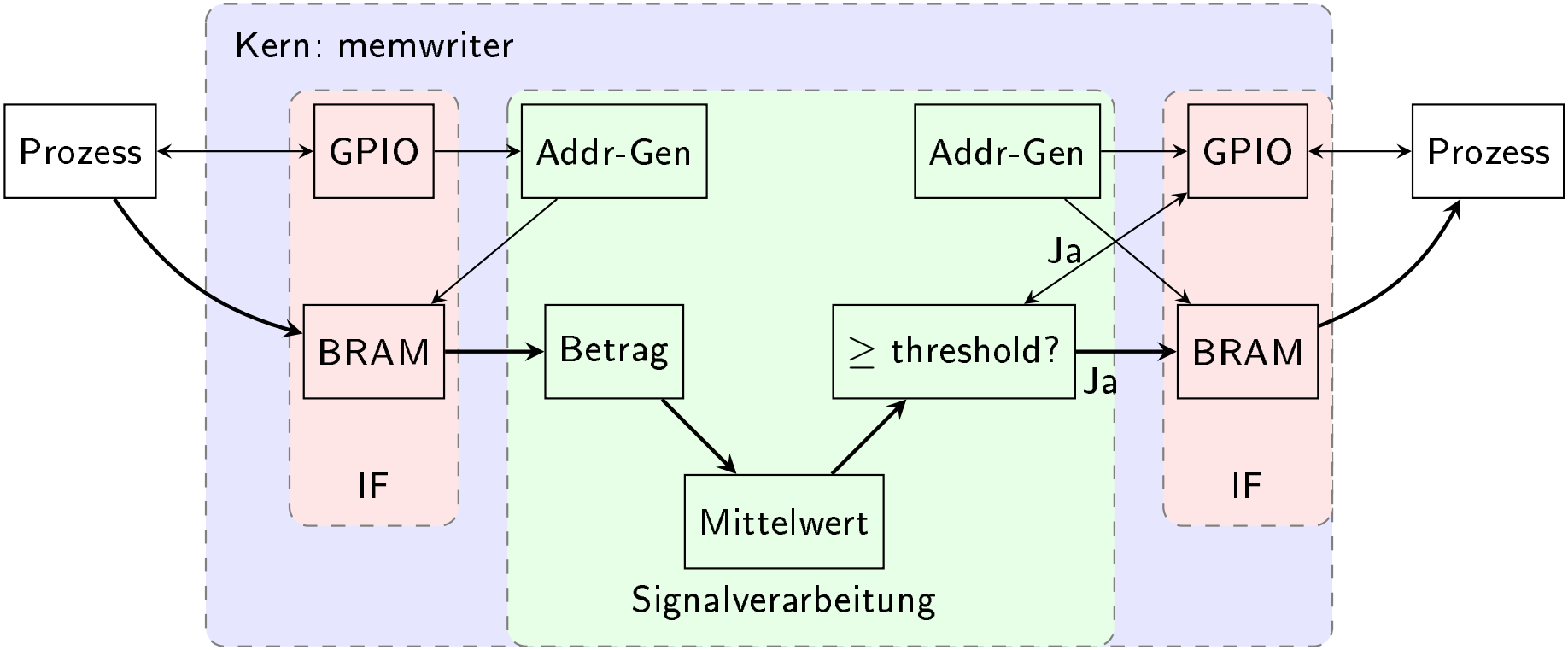

Abbildung: SV-Kern

Schwellwerterkennung: Implementierung

- Einlesen eines Stereosignals
- Splitten des Stereosignal, dann doppelte Ausführung der Module:
  - Betragsbildung
  - Mittelwertbildung über Fenster festgelegter Grösse
  - bei Überschreitung des Schwellwerts Auslösen der Übertragung über die GPIO-Steuerung

Empfangsteil

- Empfang ist auf einem Standard Linux-PC realisiert
- netcat (nc) liest Pakete von Socket und übergibt diese an die Standardeingabe von sox (Sound Exchange)
- sox konvertiert Samples in geeignetes Format und schreibt diese auf die Soundkarte, das sieht dann so aus:

```
$ nc -u -l -p 8000 | \
  sox -t raw -c 2 -r 24000 -w -s -x - \
  -t ossdsp -w -s /dev/dsp
```

System-Konfiguration über Netzwerk

- Ähnliches Vorgehen für die Kommunikation in Rückrichtung
- Netzwerkthread auf Prozessor kann über GPIO Parameter im SV-Kern setzen
- netcat kann auch zum Setzen dieser Parameter (Schwellwert, Aktivierungsdauer) benutzt werden

```
$ echo /t,100 | nc -u 192.168.1.11 8000
```

Statische und dynamische Systemteile

- Der statische hardware-abhängige Teil des System abstrahiert bzw. vereinheitlicht den Zugriff auf die Resourcen, ähnlich einem Betriebssystem.
- Der veränderbare Teil (SV-Kern) ist über eine hardware-unahängige Schnittstelle mit dem statischen Teil verbunden.
- Siehe Abb. nächste Seite

Statische und dynamische Systemteile allgemein

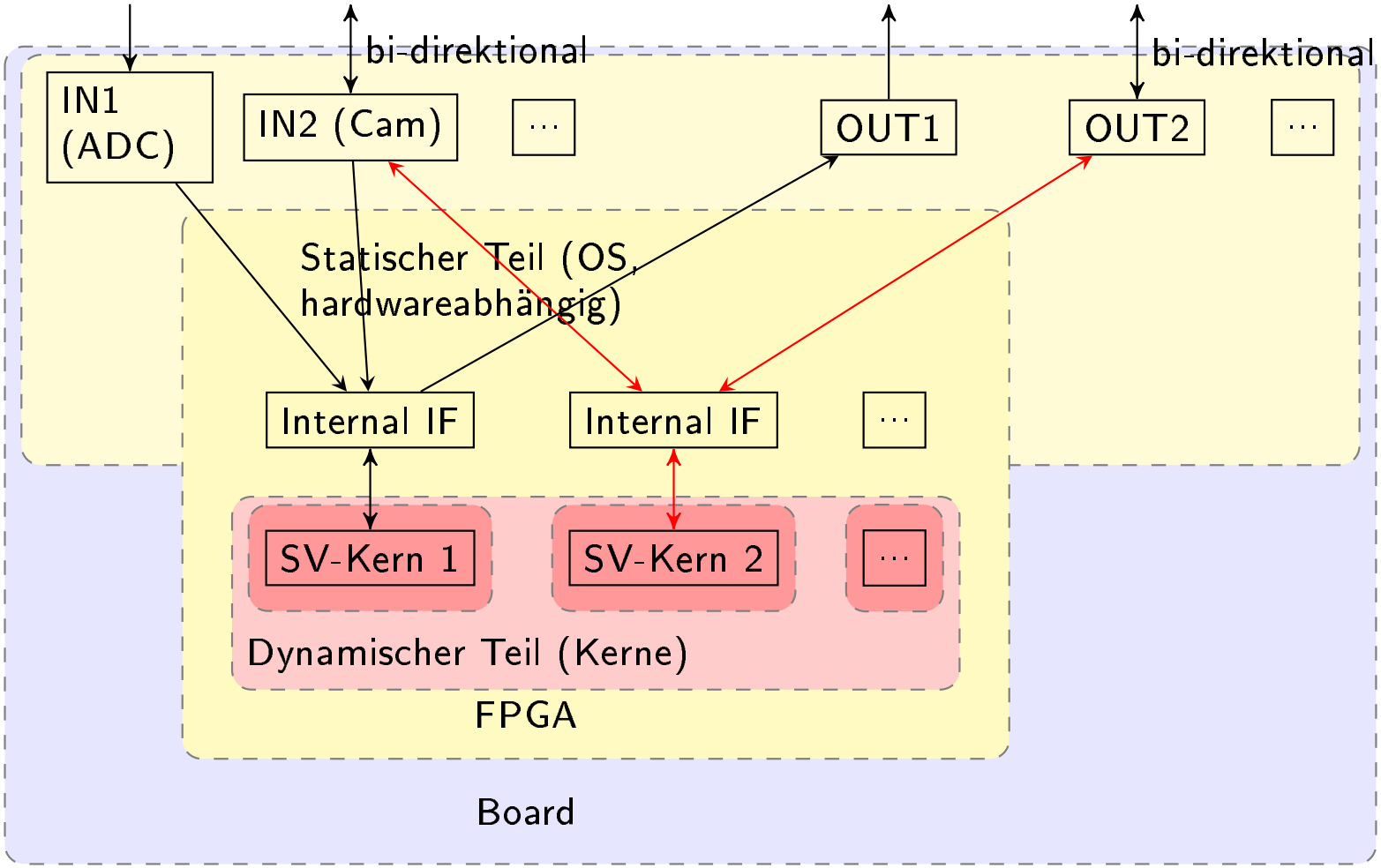

Abbildung: statischer und veränderbarer Systemteil

Statisch / dynamisch im konkreten System

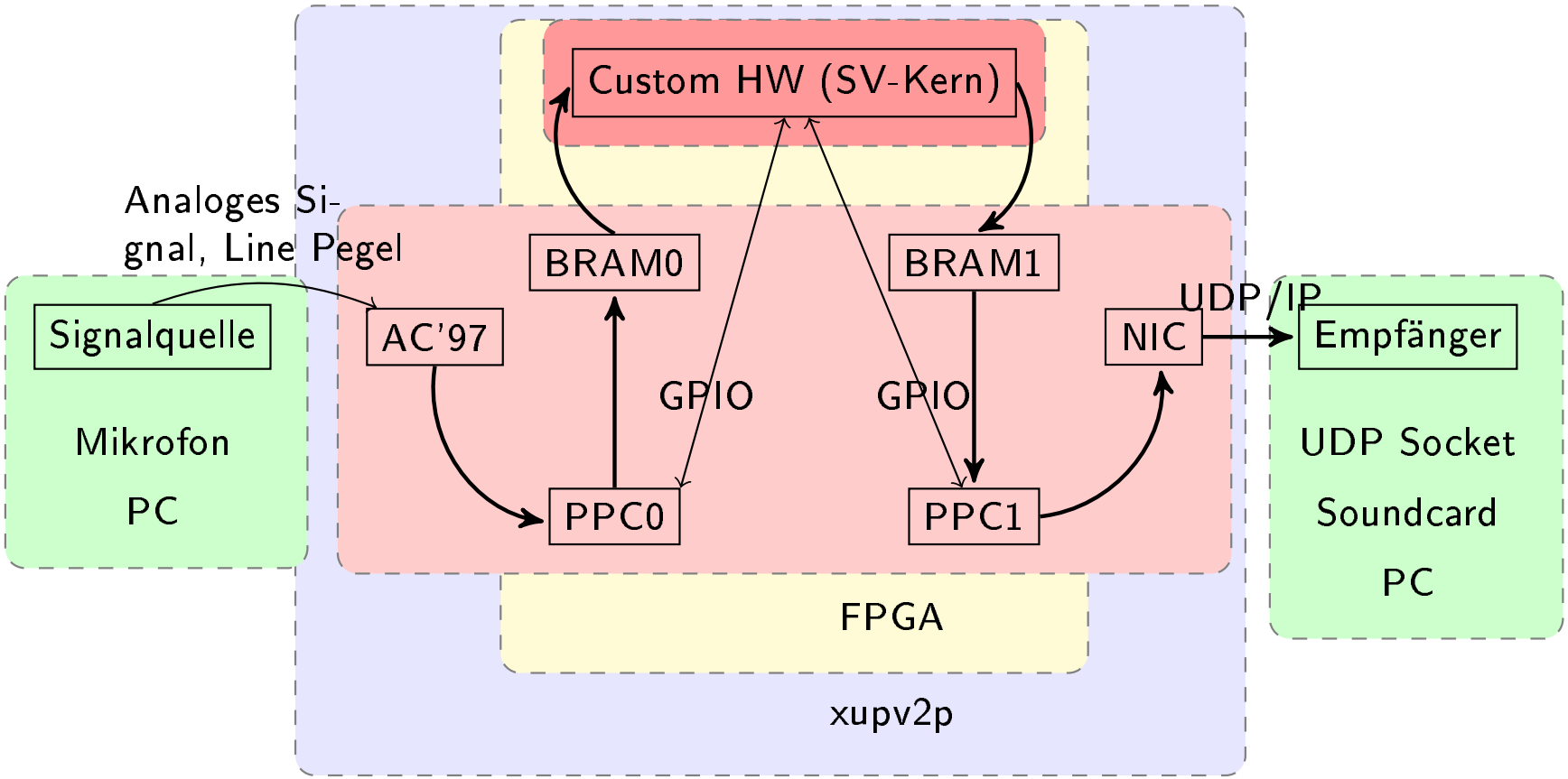

Abbildung: statischer und veränderbarer Systemteil

Weitere Schritte

- Viele Verbesserungen denkbar und möglich
- Einkanaliges Modell mit konfigurierbarer Kanalanzahl
- heterogene Kanäle
- Kompression in der Übertragung
- Redundanz durch prädiktive Signalkomponenten
- vollwertiger Rückkanal
- Entwicklung weiterer Kerne

Kerne

- Einsetzen verschiedener SV-Kerne, z.B.:
  - Filterung
  - Filterbank oder FFT Block: Frequenz-abhängige Triggerung
  - weitere Verfahren der Klassifikation: Erkennen bestimmter Wörter, Stimm- und Geräusch-Signaturen (AKF, HMM, ESN, ...)
  - Kompression
  - ...

Entwicklungsmethoden

zur Erstellung von SV-Kernen

- manuell klassischer Entwurf in VHDL
- System Generator
- XML: freesp
- Scilab / Scicos-HDL
- Evolutionäre Verfahren auf analoger oder logischer Ebene zusammen mit (D)PR-Setup

Ende

Nix mehr.