Heinrich C. Mayr, Martin Pinzger (Hrsg.): INFORMATIK 2016,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2016 1565

Einsatz von dynamisch rekonfigurierbaren FPGAs in Fahrzeugen

Peter Brungs[1] und Marcel Baunach[2]

Abstract: Die Anforderungen zukünftiger Fahrerassistenzsysteme (ADAS) an Konnektivität, Flexibilität und Verarbeitungsleistung werden immer höher, sodass aktuelle Prozessoren (MCU/DSP) durch immer aufwendigere Algorithmen an ihre Grenzen stoßen. Die vorgestellte Arbeit zeigt ein Konzept, das durch eine flexible FPGA-Architektur eine rekonfigurierbare Hardwarebeschleunigung erlaubt. Statt komplexe Algorithmen weiterhin als Software für MCUs oder DSPs zu implementieren, werden partiell rekonfigurierbare Hardwarekomponenten in FPGAs realisiert. In verteilten Systemen dieser Art werden Daten über Kommunikationsschnittstellen mit anderen Hardwarekomponenten ausgetauscht, sodass eine flexible und skalierbare Architektur entsteht. Zusätzlich gestattet das vorgestellte Konzept Spezialisierung und Redundanz kritischer Verarbeitungseinheiten hinsichtlich Performanz und Ausfallsicherheit. Die FPGA-Implementierung ist modular aufgebaut, sodass die Komponenten beliebig austauschbar und erweiterbar sind. Dies ermöglicht u.a. den Wechsel der Kommunikationsschnittstelle (z.B. von Ethernet nach CAN-Bus) oder die Erweiterung des Netzwerkprotokolls für zukünftige Anforderungen.

Keywords: Dynamische Rekonfiguration, Verteilte Systeme, FPGA, ADAS

1 Einleitung

Aufgrund wachsender Anforderungen an die Leistungsfähigkeit und Konnektivität eingebetteter Systeme bzw. elektronischer Steuergeräte (ECUs) in Fahrzeugen wurden mit AUTOSAR4 Multi-Core- und Ethernetfähigkeiten eingeführt [We13]. Zusammen mit zusätzlicher Sensorik und Aktuatorik erzeugen diese jedoch große Datenmengen, die gefiltert, aggregiert, analysiert und aufbereitet werden müssen. Hier kann rekonfigurierbare Hardware (FPGAs) aufgrund ihrer flexiblen und konfigurierbaren Architektur zur applikationsspezifischen Verarbeitung großer Datenmengen eingesetzt werden. Verglichen mit reinen Softwarelösungen bietet dies zusätzliche Möglichkeiten, wie Hardwareparallelität und Time-Multiplexing von Algorithmen, für die Ausführung hart echtzeitkritischer Berechnungen [FF12]. Zudem können durch den Einsatz von FPGAs in ECUs erkannte Defekte und Implementierungsfehler der Hardware durch erneute Konfiguration behoben werden; ggf. sogar so, dass sich das gesamte verteilte System eines Fahrzeugs selbständig reparieren kann (Configuration Scrubbing). Ein anderer Aspekt ist die Stabilität im Fehlerfall, wenn ganze ECUs ausfallen. Es besteht hier die Möglichkeit, Funktionen automatisch aus einem FPGA in ein anderes FPGA zu verlagern, sodass defekte Steuergeräte im Fahrzeugnetzwerk dynamisch ersetzt werden können. Hier will das vorliegende Papier einen neuen Ansatz für eine verteilte FPGA Architektur in Fahrzeugen aufzeigen.

[1] Universität Würzburg, Lehrstuhl für Technische Informatik, mail@peter-brungs.eu
[2] Technische Universität Graz, Institut für Technische Informatik, baunach@tugraz.at

Die Arbeit

- will eine FPGA Architektur vorstellen, die es erlaubt, heterogene sowie flexibel skalierbare FPGA-Netzwerke aufzubauen.
- zeigt exemplarisch eine modulare FPGA-Implementierung, die den Austausch und die Erweiterung von Hardwarekomponenten sowie deren Schnittstellen ermöglicht.
- zeigt, wie sich zusätzliche, rekonfigurierbare Hardwarekomponenten über ein neues Netzwerkprotokoll dynamisch in ein FPGA programmieren lassen.

Dieses Papier ist wie folgt strukturiert: Kapitel 2 gibt eine Übersicht zu vergleichbaren Architekturen, die dynamisch partielle Rekonfiguration einsetzen. In Kapitel 3 wird die Struktur der FPGA-Architektur beschrieben und in Kapitel 4 wird das eigene Konzept der netzwerkkontrollierten FPGA-Rekonfiguration vorgestellt. In Kapitel 5 wird der Ressourcenverbrauch auf dem FPGA diskutiert, und in Kapitel 6 wird mit einem Ausblick auf zukünftige Möglichkeiten die Arbeit abgeschlossen.

2 Architekturen im Vergleich

Von der Vielzahl an Ansätzen, rekonfigurierbare FPGA-Systeme zu realisieren, sollen hier einige exemplarisch aufgezeigt werden. Die Erlangen Slot Machine (ESM) [An10] ist ein Beispiel, in dem ein PowerPC Prozessor die Konfiguration steuert. Diese CPU ist außerhalb des eingesetzten Virtex II FPGAs (MotherBoard) angeordnet und steuert über eine Crossbar mit Memory Mapped I/O den Rekonfigurationsmanager. Der Rekonfigurationsmanager ist ebenfalls ein FPGA (Spartan II), der die Bitstream Informationen aus seinem Flash Speicher über eine Crossbar in die rekonfigurierbaren Slots des FPGAs lädt. Die Slots sind wiederum unterteilt in MicroSlots, die dann die kleinsten rekonfigurierbaren Einheiten der Erlangen Slot Machine bilden. In dieser Architektur ist der Ressourcenaufwand an elektronischen Komponenten sehr hoch, da zwei FPGA Bausteine sowie eine separate CPU verwendet werden. Die Anbindung zur Außenwelt ist über USB, Ethernet, IEEE1394 und PCMCIA Schnittstellen möglich.

Der Prozessor für die FPGA-Steuerung muss jedoch nicht zwingend außerhalb des FPGAs untergebracht sein. Heutige FPGAs sind ausreichend leistungsfähig, um Softcore-Prozessoren (z.B. PowerPC oder MicroBlaze) direkt integrieren zu können. Die in [Hü10] vorgestellte Lösung synthetisiert neben einem Prozessor, einen ProcessorBus, einen Speichercontroller und einen Hardware Controller für den Internal Configuration Access Port (ICAP) direkt für den FPGA-Baustein. Der Speichercontroller ist mit einem externen Speicher für die Konfigurationsdaten verbunden. Diese Lösung ist im Vergleich zur ESM hinsichtlich der Systemarchitektur effizienter, da die wichtigsten Elemente in einem einzigen FPGA zusammengefasst wurden. Ersatzweise werden nun allerdings Ressourcen in Form von Logikzellen auf dem FPGA benötigt.

Die in [BY08] gezeigte Architektur kommt komplett ohne dedizierten Prozessor aus. Dieser wurde durch einen Parallel Configuration Access Port (PCAP) ersetzt, welcher die

partielle Rekonfiguration durch den SelectMap Port, einem externen ICAP Port, steuert. Die Konfiguration-BitStreams werden im internen BlockRAM des FPGAs gespeichert. Diese Lösung ist sehr ressourcensparend und nur die nötigsten Funktionen werden für die Steuerung des SelectMap Ports realisiert. Nachteilig ist, dass die Konfigurationsinformationen fix im BlockRAM gespeichert sind und nach der FPGA-Initialisierung nicht mehr verändert werden können.

Die in [Li11] vorgestellte Architektur beschreibt ein FPGA Communication Framework (CF), basierend auf dem Ethernet Standard und der Netzwerk Socket Technologie. Diese Lösung nutzt ein eigenes FPGA Communication Protocol (FCP), basierend auf dem Netzwerkprotokoll UDP. Auf dem FPGA wird FCP durch einen Paket-Prozessor realisiert, der die Steuerung des Netzwerkverkehrs übernimmt. ICAP ist hier durch Channel Interfaces angebunden. Diese basieren auf LocalLinks, kombiniert mit Xilinx Ethernet MAC to LocalLink Interfaces und Ethernet MAC Wrapper, die teilweise Lizenzgebühren unterliegen.

3 Die Struktur der FPGA Architektur

Im Vergleich zu den vorgenannten Realisierungen basiert die in diesem Papier vorgeschlagene Lösung auf einem Ethernet MAC Controller IP von OpenCores [Mo11], der unter der LGPL Lizenz verfügbar ist. Unter [Mo09] steht ebenfalls ein CAN-Bus Controller bereit, der anstatt des Ethernet MAC Controllers integriert werden kann. Daneben werden freie Xilinx Core Module wie FIFOs und eine PLL IP Core zur Generierung unterschiedlicher Takte für die Netzwerkkomponente und der ICAP Schnittstelle verwendet. Das Netzwerkprotokoll ist in der präsentierten Arbeit beschränkt auf die IP Ebene, sodass der OSI Netzwerkstack in der Implementierung schlanker ausfällt, da auf die Funktionalität der TCP/UDP Sockets verzichtet wird. Die Steuerung übernimmt eine Finite State Machine (FSM), welche die Netzwerk- und ICAP-Schnittstelle kontrolliert, sodass die Anzahl der Komponenten minimiert werden konnte. Über die Ethernet-Schnittstelle werden die Steuer-, Konfigurations- und Ergebnisdaten übertragen, sodass auch kleine FPGA Implementierungen ermöglicht werden.

Klassische Vorgehensweisen zur Programmierung des FPGA-Konfigurationsspeichers benötigen eine direkte Punkt-zu-Punkt Verbindung zwischen PC und FPGA, z.B. über USB oder PCI-Bus. Die hier gezeigte Lösung erlaubt es, die Rekonfiguration über Netzwerke wie Ethernet oder CAN-Bus aus der Ferne durchzuführen, sodass FPGA und Programmierer räumlich getrennt sein können.

Abbildung 1 zeigt den grundsätzlichen Aufbau der vorgestellten FPGA Architektur: Das Netzwerkinterface XETHERNET empfängt die Daten aus dem Netzwerk und sendet diese, kontrolliert durch XETHERNET_CONTROL, an die Reconfiguration Unit ICAP_SP6. Die Reconfiguration Unit hat eine direkte Verbindung in die Dynamic Reconfiguration Area, in die letztlich alle synthetisierten Hardwarekomponenten programmiert werden. Daten und Verarbeitungsergebnisse werden ebenfalls von der ICAP_SP6 Komponente aus der Dynamic Reconfiguration Area gelesen und durch das XETHERNET Modul an das externe Netzwerk gesendet.

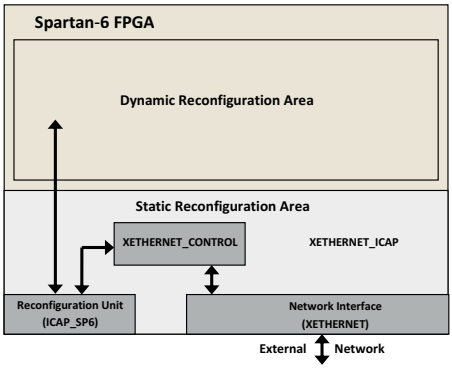

Abb. 1: Grundsätzliche Struktur des dynamisch rekonfigurierbaren FPGA

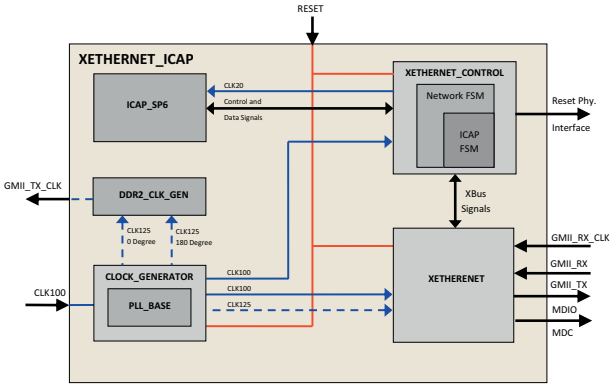

Abb. 2: XETHERNET_ICAP Komponente

Abbildung 2 zeigt den Aufbau der XETHERNET_ICAP Komponente zur Steuerung des FPGA Konfigurationsspeichers. Diese besteht aus vier Hauptkomponenten: ICAP_SP6 bildet die Schnittstelle zum Spartan-6 Konfigurationsspeicher und XETHERNET die Schnittstelle zum Netzwerk. XETHERNET_CONTROL überwacht beide vorgenannten Komponenten und steuert deren Aktionen durch die Netzwerk FSM und die ICAP FSM. Der CLOCK_GENERATOR generiert die notwendigen Takte für ICAP_SP6, XETHERNET und XETHERNET_CONTROL. Diese Architektur braucht im Gegensatz zu den prozessorgetriebenen Lösungen weniger Ressourcen und beschränkt sich auf die nötigsten Komponenten, um die Transferlast von Konfigurationsdaten zwischen FPGA und den Host-Anwendungen gering zu halten.

Abbildung 3 zeigt die XETHERNET Komponente im Detail. Als Basis für die Gigabit-Ethernet Realisierung dienen der GEMAC Controller [Et09] und Informationen über ethmac aus [Mo11]. Diese Lösung ermöglicht Herstellerunabhängigkeit mit eventuell anfallenden Lizenzkosten. Die Taktfrequenz des XETHERNET Komponente ist 100 MHz und die der Ethernet Schnittstelle liegt bei 125 MHz im GBit Modus. XETHER_Transmitter und XETHER_Receiver dienen in der GEMACFIFO Komponente als Schnittstelle zwi-

schen diesen beiden Frequenzbereichen. Die REGISTER Komponente dient der Steuerung und der Konfiguration der XETHERNET Komponente. Der modulare Aufbau der vorgestellten Architektur ermöglicht den Austausch der Kommunikationsschnittstelle durch Alternativen, wie z.B. durch einen CAN-Bus Controller anstatt des GEMAC Controllers [Mo09].

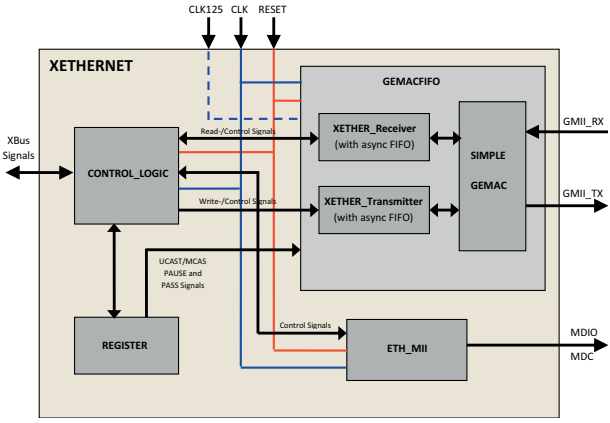

Abb. 3: XETHERNET Komponente

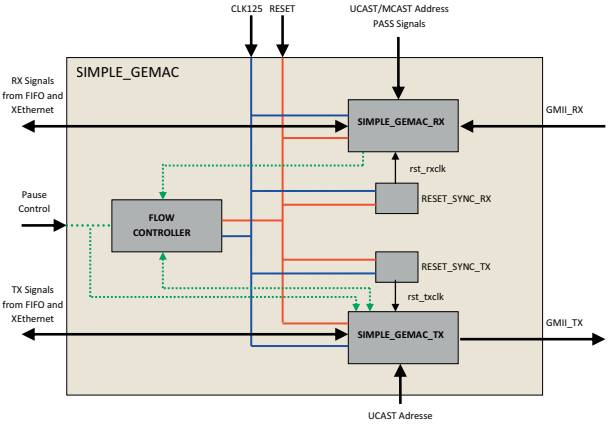

Abb. 4: SIMPLE_GEMAC Komponente

Abbildung 4 zeigt die SIMPLE_GEMAC Komponente. Diese wurde für die XETHERNET Komponente übernommen, da sie in anderen Projekten bereits ihre Funktionalität bewiesen hatte [Et09] [Wi16]. SIMPLE_GEMAC setzt sich im Wesentlichen aus einer Empfangs- und einer Sendeeinheit zusammen, die den Datenfluss steuern, d.h. Pausen-Pakete und Multi- oder UniCast-Signale empfangen und versenden, die Prüfsummen erzeugen und den Vorspann eines Ethernets-Paketes generieren.

4 Das Konzept der netzwerkkontrollierten FPGA-Rekonfiguration

4.1 Die Rekonfiguration des Spartan-6

Das Spartan-6 FPGA lässt sich dynamisch, d.h. im laufenden Betrieb, rekonfigurieren [Se06]. Im Gegensatz zu den Virtex FPGAs, bei denen die Partitionen direkt beschrieben werden können, wird bei den Spartan-6 FPGAs das "Differencing-Based Partial Reconfiguration"[Et07] Verfahren verwendet, d.h. die Rekonfigurationsinformationen werden durch Differenzbildung zweier Bitstreams generiert. Die Befehlssequenz im Bitstream wird dann in die interne ICAP Schnittstelle geladen. Der Vorteil der Differencing-Base Methode ist, dass während der Programmierung der Teil des FPGAs weiter arbeitet, der nicht geändert werden soll. Die Schnittstellen zwischen dem nicht konfigurierten und dem konfigurierten Teil sind nach der Rekonfiguration undefiniert und müssen z.B. durch ein Reset-Signal in einen definierten Zustand gebracht werden [UG15b].

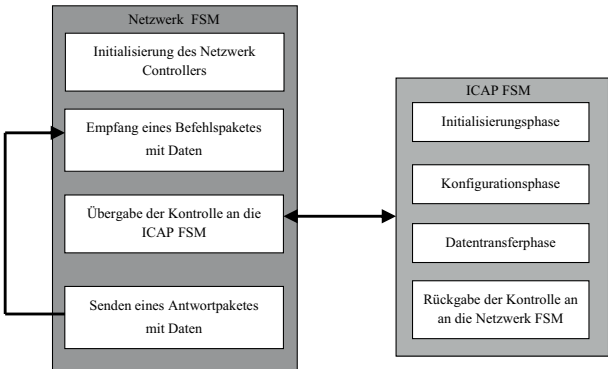

Abb. 5: Innere Struktur der XETHERNET_CONTROL Komponente

4.2 Die Rekonfigurations-Steuereinheit

Die Netzwerk FSM (Abbildung 5) hat vier übergeordnete Zustände: Initialisierung des Netzwerkkontrollers, Empfang von Befehlen/Daten, Übergabe der Kontrolle an die ICAP FSM und Senden von Antwortpaketen/Daten. Diese Zustände können je nach Bedarf unterschiedlich ausgeprägt werden, z.B. benötigt eine Ethernet Schnittstelle eine andere Initialisierung als eine CAN-Bus Schnittstelle. Die ICAP FSM folgt einer festen Abfolge von Schritten, um die Daten in den Rekonfigurationspeicher des FPGAs zu schreiben.

4.3 Steuerung der ICAP FSM

Die Steuerung der ICAP FSM orientiert sich an der fest definierten Struktur des Konfigurations-Bitstreams [UG15a] aus Tabelle 1:

Block	Beschreibung
DUMMY	16 Dummy Worte für BPI Address shift cycle
SYNC	Zwei Synchronisationsworte
HEADER	Initialisierung der Konfigurationsregister
CFG BODY	Start Adresse Schreib/Lese-Befehle FDRI/FDRO Inhalte des FPGA Konfigurationsspeichers AUTO CRC Worte
HEADER2	Initialisierung der Konfigurationsregister für die nächste Operation
DESYNC	Desynchronisierungsbefehl

Tab. 1: Bitstream Struktur des Spartan-6

Die Bitstream-Sequenzen lassen sich in standardisierte Blöcke gruppieren, sodass diese für verschiedene Einsatzzwecke, wie Lese- oder Schreiboperationen, wiederverwendet werden können. In der VHDL-Realisierung korrespondieren diese Blöcke mit den Zuständen der ICAP FSM. Abbildung 6 zeigt die derzeit definierten Abläufe:

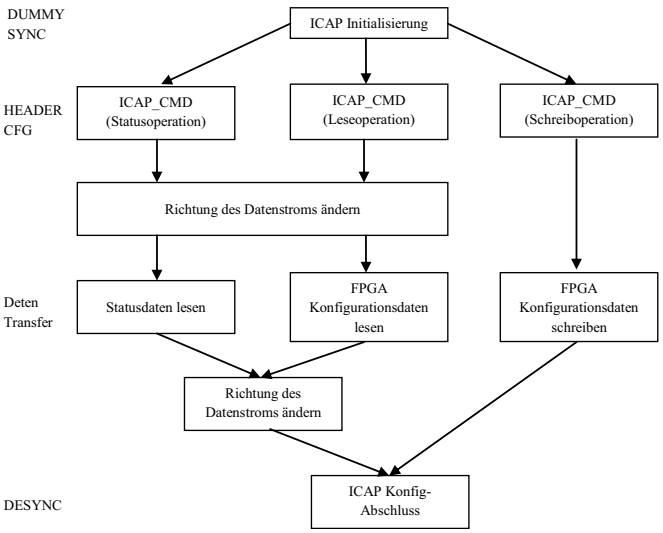

Abb. 6: Ablaufdiagramm der ICAP FSM Steuerung

1. **Konfigurations-Ablauf:** Der Zugriff auf das Konfigurationsregister erlaubt es, den Zustand und die Kennung eines FPGAs zu ermitteln [UG15a]. Die Netzwerk FSM empfängt durch die XETHERNET Komponente den Befehl, die Konfigurationsdaten zu ändern. Dies bewirkt, dass die ICAP FSM auf dem FPGA startet und die Blöcke im Pfad der Statusoperationen bearbeitet.

2. **Lese-Ablauf:** Die Leseoperation geschieht analog zum Konfigurations-Ablauf. Der Unterschied liegt im Detail, da nun der Konfigurationsspeicher anstatt der Statusregister gelesen wird. Der Ablauf ist im Pfad der Leseoperationen verdeutlicht.

3. **Schreib-Ablauf:** Bei der Schreiboperation ist die Anzahl der Blöcke im Pfad kleiner, sodass die ICAP-FSM in diesem Ablauf mit weniger Zuständen auskommt.

4. **Test-Ablauf:** Zurzeit existieren zwei Test-Abläufe. Mit dem ICMP-Ping oder dem ICAP-Ping testet man, ob das FPGA ansprechbar ist und die Netzwerk FSM aktiv ist. Der ICMP-Ping reagiert auf Pings von einem Kommandozeilen Werkzeug und der ICAP-Ping wird von einer Software-API heraus aufgerufen. Diese generiert ein ICAP-Netzwerkpaket, das während der Initialisierungsphase die Funktionsfähigkeit der FSM testet.

4.4 Steuerung der Netzwerk FSM durch ein Netzwerkprotokoll

Das Netzwerkprotokoll zwischen Host und FPGA arbeitet auf der IP Schicht des OSI-Referenzmodells und insbesondere auf der ICMP-Protokoll Ebene. Der Grund für diese Auswahl ist, dass man sich immer noch innerhalb eines weitverbreiteten Netzwerkstandards bewegt, sodass handelsübliche Geräte wie Router, Switches und Hub zur Kommunikation zwischen Host und FPGA oder zwischen FPGA und FPGA verwenden werden können, da die Adressierung über die IP- und MAC-Adresse des FPGAs stattfindet. Die nächsthöhere Protokollschicht im OSI-Referenzmodell [SZ14] wäre die TCP/UDP Schicht, die zwar mehr Funktionalität, z.B. Sockets, bietet, für die vorliegende Anwendung jedoch nicht nötig ist und nur zusätzliche Hardwareressourcen innerhalb des FPGAs zur Verarbeitung des Protokolls bindet.

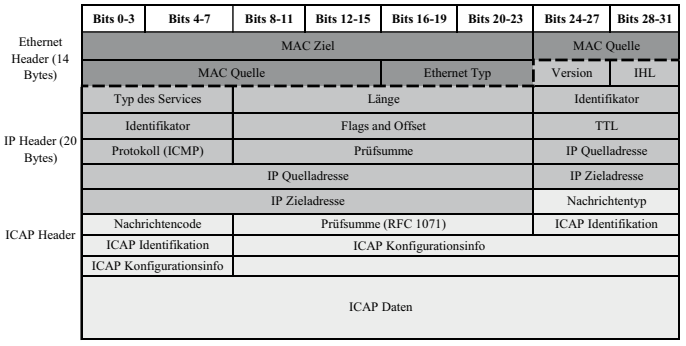

Abb. 7: ICAP Netzwerkprotokoll Headerstruktur

Abbildung 7 zeigt die Schichten anhand der Headerstruktur des ICAP Protokolls. Das ICAP Protokoll besteht aus zwei Schichten, die zum Data Link Layer und zum Network Link Layer des OSI-Referenzmodells korrespondieren [SZ14]. Im Network Link Layer benutzt das ICAP Protokoll nicht zugewiesene ICMP Nachrichtentypen und Nachrichtencodes des ICMP Standards [Po81]. Das ICAP und das ICMP Protokoll sind Teil des IP Protokolls und können aus diesem Grund die Paket Routing Funktionalität des IP Headers nutzen.

Die neuen Nachrichtencodes und -typen für Kommandos (Request) und Antworten (Reply) sind in Abbildung 8 und Abbildung 9 gezeigt. Zum Vergleich ist die ICMP-Ping Funktion zusammen mit den neuen ICAP Funktionen dargestellt.

	Typ	Code	Prüfsumme (16 Bit)	Identifikation (16 Bit)	Konfigurationsinfo		Daten
ICMP-Request-Ping	0x08	0x00	Wert	Wert	Sequenz-Nummer	Füllwerte	
ICAP-Request-Ping	0x18	0x00	Wert	Wert	Sequenz-Nummer	Füllwerte	
ICAP-Request-ReadConfig	0x18	0x01	Wert	Wert	RegisterNummer		
ICAP-Request-Status	0x18	0x02	Wert	Wert			
ICAP-Request-IDCODE	0x18	0x03	Wert	Wert			
ICAP-Request-Read	0x18	0x04	Wert	Wert	Major	Minor	
ICAP-Request-Write	0x18	0x05	Wert	Wert	Major	Minor	Data
ICAP-Request-WriteConfig	0x18	0x06	Wert	Wert	RegisterNummer	1. Wert	2-n. Wert

Abb. 8: Struktur des ICAP Header für Request Netzwerkpakete

	Type	Code	Prüfsumme (16 Bit)	Identifikation (16 Bit)	Konfigurationsinfo		Daten
ICMP-Reply-Ping	0x00	0x00	Wert	Wert	Sequenz-Nummer	Füllwerte	
ICAP-Reply-Ping	0x19	0x00	Wert	Wert	Sequenz-Nummer	Füllwerte	
ICAP-Reply-Config	0x19	0x01	Wert	Wert	RegisterNummer	1. Wert	2-n. Wert
ICAP-Reply-Status	0x19	0x02	Wert	Wert	1. Wert	2. Wert	3-n. Wert
ICAP-Reply-IDCODE	0x19	0x03	Wert	Wert	1. Wert	2. Wert	
ICAP-Reply-Read	0x19	0x04	Wert	Wert	Major	Minor	Data
ICAP-Reply-Write	0x19	0x05	Wert	Wert	Major	Minor	Status
ICAP-Reply-WriteConfig	0x19	0x06	Wert	Wert	RegisterNummer	Status	

Abb. 9: Struktur des ICAP Header für Reply Netzwerkpakete

Auf der Hostseite wird die Kommunikation zurzeit mit einer in C geschriebenen Software durchgeführt. Die Netzwerkkommunikation basiert auf Socket und PCAP Libraries von Windows und Linux. Auf der FPGA Seite wird die Steuerung zurzeit durch eine FSM durchgeführt. Die Quelldateien sind in VHDL implementiert.

5 Evaluierung der XETHERNET_ICAP Schnittstelle

Der FPGA Ressourcenverbrauch des vorgestellten Ansatzes ist in Tabelle 2 zusammengefasst. Die logischen Elemente (LUT) verbrauchen etwa 6% der verfügbaren Gesamtressourcen. Die IOs liegen bei 12% und die RAMB16 (18kb Block RAM) bei etwa 3%.

Typ der Logik	Verfügbar	Anzahl	Verbrauch in %
Anzahl der Slice Register	54.576	1.364	2%
Anzahl der Slice LUTs	27.288	1.746	6%
Verbrauchter Speicher	6.408	46	1%
Anzahl der IOs	218	27	12%
Anzahl der RAMB16	116	3	3%

Tab. 2: Ressourcenverbrauch auf einem Spartan-6-slx45 FPGA

Abbildung 10 zeigt, wie die statische Logik in einem engen Bereich des FPGAs konzentriert ist. Dies wurde durch entsprechende Vorgaben während der Synthese erreicht. Ziel war es, die Logik in der Nähe der physikalischen Hardwareschnittstellen des Spartan-6 FPGA anzuordnen, um lange Signalwege und das Kreuzen von Signalen aus statischen und dynamischen Komponenten zu minimieren. Die Netzwerk I/O-Ports sind rechts unten von der Steuerlogik und die ICAP Komponente links unten davon untergebracht.

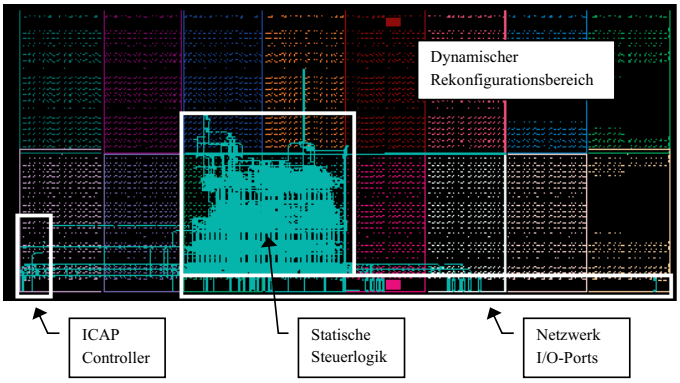

Abb. 10: Logik- und Schnittstellenverteilung auf dem FPGA

Durch die in Abbildung 10 gezeigte Anordnung der statischen Logik, lassen sich große freie Flächen für die dynamische Rekonfiguration auf der oberen FPGA-Hälfte schaffen. Die ICAP-Schnittstelle hat im Spartan-6 eine Arbeitsfrequenz von 20 MHz bei einer Datenbusbreite von 16 Bit. Eine Konfigurationsdatei enthält für diesen FPGA-Typ 1.484.525 Bytes, sodass in ca. 37 ms ein komplettes FPGA rekonfiguriert werden kann. Ein einzelner Frame hat eine Größe von 130 Bytes, sodass hier die Programmierung 3.25μs dauert. Die eingesetzte Ethernet Schnittstelle hat einen Arbeitstakt von 125 MHz bei 8 Bit Datenbreite, sodass ein Konfigurationsframe weniger als 1μs zur Übertragung in die XETHERNET_ICAP Komponente benötigt.

6 Zusammenfassung und Ausblick

Das in diesem Papier präsentierte Konzept zeigt wie synthetisierte Hardware ”remote” über gängige Netzwerkschnittstellen dynamisch und im laufenden Betrieb in ein FPGA konfiguriert werden kann. Die bisher erforderliche Punkt-zu-Punkt Verbindung zwischen FPGA und Host entfällt. Vergleichbar mit der Aktualisierung von Software für klassische MCU/DSP, erlaubt dies insbesondere die kosteneffiziente und nachträgliche Optimierung, Erweiterung und Fehlerbehebung synthetisierter Hardware in komplexen vernetzten Systemen (z.B. ADAS in Fahrzeugen). Diverse Netzwerkschnittstellen können durch die flexible und modulare Struktur integriert werden. Hier sind im ADAS-Umfeld (u.a.) FlexRay, CAN, LIN, und Ethernet im Gebrauch.

6.1 Flexible Struktur

Die XETHERNET_ICAP Komponente ist modular entworfen und könnte neben dem Ethernet Controller oder dem CAN-Bus Controller auch andere Netzwerktechnologien innerhalb des XETHERNET Moduls einbinden. Das XETHERNET Modul benutzt eine einfache Busverbindung zur XETHERNET_CONTROL Komponente, sodass es sich für andere Anwendungen mit wenig Aufwand anpassen lässt.

6.2 Sicherheitsaspekte

Die FPGA Struktur ist aufgeteilt in einen dynamischen und statischen Teil (s. Abbildung 1 und Abbildung 10). Die statische Logik lässt sich bei der Initialisierung des FPGAs aus einem lokalen Flash-Speicher laden, der über das Netzwerk nicht erreichbar ist. Dieser Speicher lässt sich jedoch über eine separate Wartungsschnittstelle aktualisieren. Die Netzwerk FSM und die ICAP FSM lassen sich so konfigurieren, dass nur Änderungen in den von außen dynamisch rekonfigurierbaren Bereichen des FPGAs möglich sind. Damit ist es nicht möglich den statischen Bereich zu überschreiben und somit die Komponenten zur Rekonfiguration zu entfernen, bzw. zu modifizieren.

Neben der operativen Sicherheit gibt es auch Sicherheitsanforderungen entsprechend dem Standard ISO26262. FPGA Hersteller wie XILINX und Altera haben bereits ihre Entwicklungswerkzeuge und Methoden dementsprechend vom TÜV zertifizieren lassen [Al13] [PB14].

6.3 Anwendungsszenarien

Die vorliegende Arbeit dient als Grundlage für die Entwicklung komplexer Systeme basierend auf vernetzter, rekonfigurierbarer Hardware. Abbildung 11 und Abbildung 12 zeigen Nutzungsmöglichkeiten im Rahmen verteilter Fahrerassistenzsysteme (ADAS). Anstelle von MCUs/DSPs übernehmen applikationsspezifisch konfigurierbare FPGAs die Datenverarbeitung und steuern sowohl Sensoren als auch Aktuatoren an.

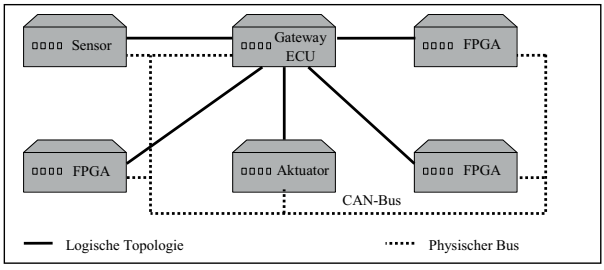

Abb. 11: Zentral kontrolliertes Netzwerk

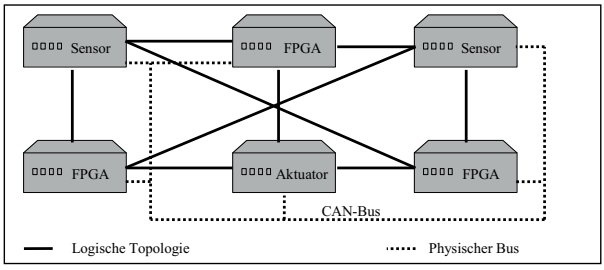

Abb. 12: Verteilt kontrolliertes Netzwerk

Im ersten Szenario (Abbildung 11) kontrolliert ein zentrales Steuergerät (ECU) sowohl die Konfigurationen der FPGAs als auch die Datenverteilung auf die entsprechend konfigurierten Verarbeitungseinheiten. Fällt eine Verarbeitungseinheit aus, wird ein anderes FPGA rekonfiguriert und die Daten flexibel zu diesem neuen Ziel umgeleitet. Das zweite Szenario (Abbildung 12) veranschaulicht die selbständige Zusammenarbeit der FPGAs ohne zentrales Steuergerät. Die Daten werden autonom von den Sensoren empfangen, im Netzwerk aufbereitet und an die Empfänger weitergeleitet. Durch ständige Selbstkontrolle wird überprüft, ob die Verarbeitungseinheiten noch einwandfrei funktionieren. Im Fehlerfall wird die Funktionalität eines defekten FPGAs durch ein anderes FPGA bereitgestellt. Dadurch wird das Gesamtsystem robust und ausfallsicher.

6.4 Ausblick

Die nächsten Schritte sind, weitere Netzwerkschnittstellen an die FPGA-Architektur anzubinden. Beispielsweise sollen mehrere CAN-Bus Module in ein FPGA integriert werden, um ein Netzwerk von selbständig arbeitenden und selbstkontrollierenden FPGAs aufzubauen. Das Kommunikationsprotokoll würde für diesen Zweck um Befehle für die Selbstkontrolle erweitert.

Aktuelle MCUs/DSPs bewegen sich mit ihrer Rechenleistung im Bereich von etwa 1.5 GFlops/Core (ARM Cortex-A15). Hingegen können die FPGAs, wie auch die aktuellen Grafikprozessoren (GPUs) Rechenleistungen im Bereich von TFlops (FP32) bieten. Die Kosten für FPGAs und GPU sind zur Zeit im Bereich um 0.30 €/GFlops [GP16]. Durch den Rechenleistungshunger von ADAS Systeme werden die FPGAs wegen ihrer Flexibilität und Leistungsfähigkeit aber immer attraktiver.

Literaturverzeichnis

[Al13] Alteras Leadership in Functional Safety Expands to ISO 26262, 2013. www.altera.com/solutions/industry/automotive/applications/safety/overview.tablet.html, Stand: 17.06.2016.

[An10] Angermeier, Josef; Bobda, Christophe; Majer, Mateusz; Teich, Jürgen: Erlangen Slot Machine: An FPGA-Based Dynamically Reconfigurable Computing Platform. In (Platzner, Marco; Teich, Jürgen; Wehn, Norbert, Hrsg.): Dynamically Reconfigurable Systems: Architectures, Design Methods and Applications. Springer Netherlands, Dordrecht, S. 51–71, 2010.

[BY08] Bayar, Salih; Yurdakul, Arda: Self-reconfiguration on Spartan-III FPGAs with compressed partial bitstreams via a parallel configuration access port (cPCAP) core. In: Research in Microelectronics and Electronics, 2008. Ph.D. S. 137–140, June 2008.

[Et07] Eto, Emi: , XAPP290, Difference-Based Partial Reconfiguration, 2007. www.xilinx.com/support/documentation/application_notes/xapp290.pdf, Stand: 11.06.2016.

[Et09] Ettus, Matt: , usrp2 (simple_gemac), 2009. www.github.com/EttusResearch/fpga/tree/master/usrp2, Stand: 11.06.2016.

[FF12] Fons, Francisco; Fons, Mariano: FPGA-based Automotive ECU Design Addresses AUTOSAR and ISO26262 Standards. Xcell Journal 12/01, S. 20–31, 2012.

[GP16] GPU vs FPGA Performance Comparison, 2016. www.bertendsp.com/pdf/whitepaper/BWP001_GPU_vs_FPGA_Performance_Comparison_v1.0.pdf, Stand: 17.06.2016.

[Hü10] Hübner, Michael; Göhringer, Diana; Noguera, Juanjo; Becker, Jürgen: Fast dynamic and partial reconfiguration data path with low hardware overhead on Xilinx FPGAs. In: Parallel Distributed Processing, Workshops and Phd Forum (IPDPSW), 2010 IEEE International Symposium on. S. 1–8, April 2010.

[Li11] Lieber, Peter Andrew: , FPGA Communication Framework for Communication, Debugging, Testing, and Rapid Prototyping, 2011. scholarsarchive.byu.edu/etd/3039, Stand: 11.06.2016.

[Mo09] Mohor, Igor: , CAN Protocol Controller, 2009. www.opencores.org/project,can, Stand: 10.05.2016.

[Mo11] Mohor, Igor: , Ethernet MAC 10/100 Mbps, 2011. www.opencores.org/project,ethmac, Stand: 10.05.2016.

[PB14] PB015, Xilinx All Programmable Functional Safety Design Flow Solution, 2014. www.xilinx.com/publications/prod_mktg/safety-guidelines.pdf, Stand: 17.06.2016.

[Po81] Postel, Jon: , Internet Control Message Protocol Specification (RFC792), 1981. tools.ietf.org/html/rfc792, Stand: 11.06.2016.

[Se06] Sedcole, Nicholas Pete: , Reconfigurable Platform-Based Design in FPGAs for Video Image Processing (Chapter 6), PHD Thesis, 2006. cas.ee.ic.ac.uk/people/nps/publications.html, Stand: 14.10.2014.

[SZ14] Spurgeon, Charles; Zimmerman, Joann: Ethernet: The Definitive Guide, Second Edition. O'Reilly Media, Inc, 2014.

[UG15a] UG380, Spartan-6 FPGA Configuration, Version 2.8, 2015. www.xilinx.com/support/documentation/user_guides/ug380.pdf, Stand: 11.06.2016.

[UG15b] UG702, Partial Reconfiguration User Guide, Version 14.1, 2015. www.xilinx.com/support/documentation/sw_manuals/xilinx14_1/ug702.pdf, Stand: 11.06.2016.

[We13] Weber, Marc: AUTOSAR lernt Ethernet. In: HANSER automotive networks. Hanser, S. 30–33, 2013.

[Wi16] Williams, Joel: , Digilent Atlys, 2016. www.joelw.id.au/FPGA/DigilentAtlysResources, Stand: 11.06.2016.