Universität Leipzig
Fakultät für Mathematik und Informatik
Institut für Informatik

Masterarbeit im Fach Informatik

Entwicklung einer freien RISC-V-Workstation mit Unterstützung für GNU/Linux auf Basis des Xilinx ML507-Entwicklungsboards

Klemens Schölhorn

6. Juni 2018

Betreut von Michael Krause

Inhaltsverzeichnis

1 Einleitung

Mit RISC-V existiert seit einigen Jahren eine vielversprechende, moderne und freie Befehlssatzarchitektur für Prozessoren. Im Gegensatz zu den etablierten Architekturen wie x86 und ARM ermöglicht eine freie Architektur eine Implementierung ohne Lizenzgebühren. Außerdem erlaubt sie Veränderungen an der Architektur selbst, was für die Weiterentwicklung von Befehlssatzarchitekturen essenziell ist. Darüber hinaus ist eine freie Befehlssatzarchitektur die Voraussetzung für die Entwicklung freier und offener Hardware. Bisher verfolgte vor allem OpenRISC [6] dieses Ziel. Doch während diese Architektur in der OpenSource-Gemeinde sehr erfolgreich ist, konnte sie bislang kaum kommerziellen Erfolg erzielen. RISC-V hat dagegen innerhalb kurzer Zeit sowohl in der Community als auch bei Unternehmen eine große Anzahl Unterstützer gefunden.

In dieser Arbeit soll eine GPL-lizenzierte RISC-V-Workstation entwickelt werden. Als Hardwareplattform dient das Xilinx ML507-Entwicklungsboard mit einem Virtex-5-FPGA, welches dem Autor im Rahmen dieser Arbeit von der Abteilung Technische Informatik der Universität Leipzig zur Verfügung gestellt wurde. Die Workstation soll selbstständig ein GNU/Linux-System booten, Eingaben von einer Tastatur entgegennehmen und Ausgaben auf einem per DVI angeschlossenen Bildschirm anzeigen. Für die einzelnen Komponenten sollen, wenn möglich, bereits vorhandene, freie Implementierungen verwendet werden. Die übrigen Komponenten sollen in VHDL implementiert werden.

Die Arbeit besteht aus vier Teilen. Im ersten Teil werden die für das Verständnis der folgenden Kapitel nötigen Grundlagen beschrieben. Anschließend wird der als Basis für die Workstation ausgewählte Rocket-Prozessor vorgestellt. Im dritten Teil wird die Entwicklung der für die Workstation nötigen Peripheriegeräte und deren Integration in das Gesamtsystem erläutert. Der letzte Teil betrachtet schließlich den Bootprozess, das verwendete Linux-System und die notwendigen Treiber für die im vorhergehenden Kapitel entwickelten Geräte.

2 Grundlagen

Als Grundlage für die Workstation dient das Entwicklungsboard ML507 für das Xilinx Virtex-5 XC5VFX70T FPGA (siehe Abbildung 2.1). Es enthält neben dem FPGA eine Vielzahl von Anschlüssen und Zusatz-ICs, von denen die meisten in dieser Arbeit jedoch nicht verwendet werden. Die LEDs und Schalter dienen bei der Implementierung der Workstation als GPIO-Pins (general purpose input/output) für den Prozessor. Als Arbeitsspeicher wird der auf dem Board verfügbare DDR2-SDRAM verwendet. Weiterhin kommen der DVI-Ausgang und der zugehörige Signalerzeugungschip bei der Implementierung eines Text-Terminals zum Einsatz. Die UART-Schnittstelle dient ebenfalls der Ausgabe und zusätzlich der Eingabe von Zeichen. Außerdem werden einige frei verfügbare Pins des FPGAs zur Ansteuerung einer SD-Karte über SPI (Serial Peripheral Interface) verwendet. Die Konfiguration des FPGAs wird von einem der integrierten Flash-Speicher oder direkt per JTAG geladen.

Das ML507-Board wird bereits seit 2014 nicht mehr verkauft [31]. Mitte 2017, während der Autor bereits mit dieser Arbeit beschäftigt war, wurden sämtliche das Board betreffende Support-Informationen und Dokumente von der Xilinx-Website entfernt. Während die meisten Textseiten und PDF-Dokumente im Internet Archive enthalten sind, fehlen die oft in zip-Archiven verpackten Beispielprojekte und Skripte.

Die nachfolgenden Abschnitte geben einen Überblick über die für die Implementierung der Workstation relevanten Protokolle und Komponenten.

2.1 DVI

Die Ausgabe des für die Workstation entwickelten Terminals erfolgt über den DVI-Ausgang des Entwicklungsboards. Das Digital Visual Interface ist ein Standard zur

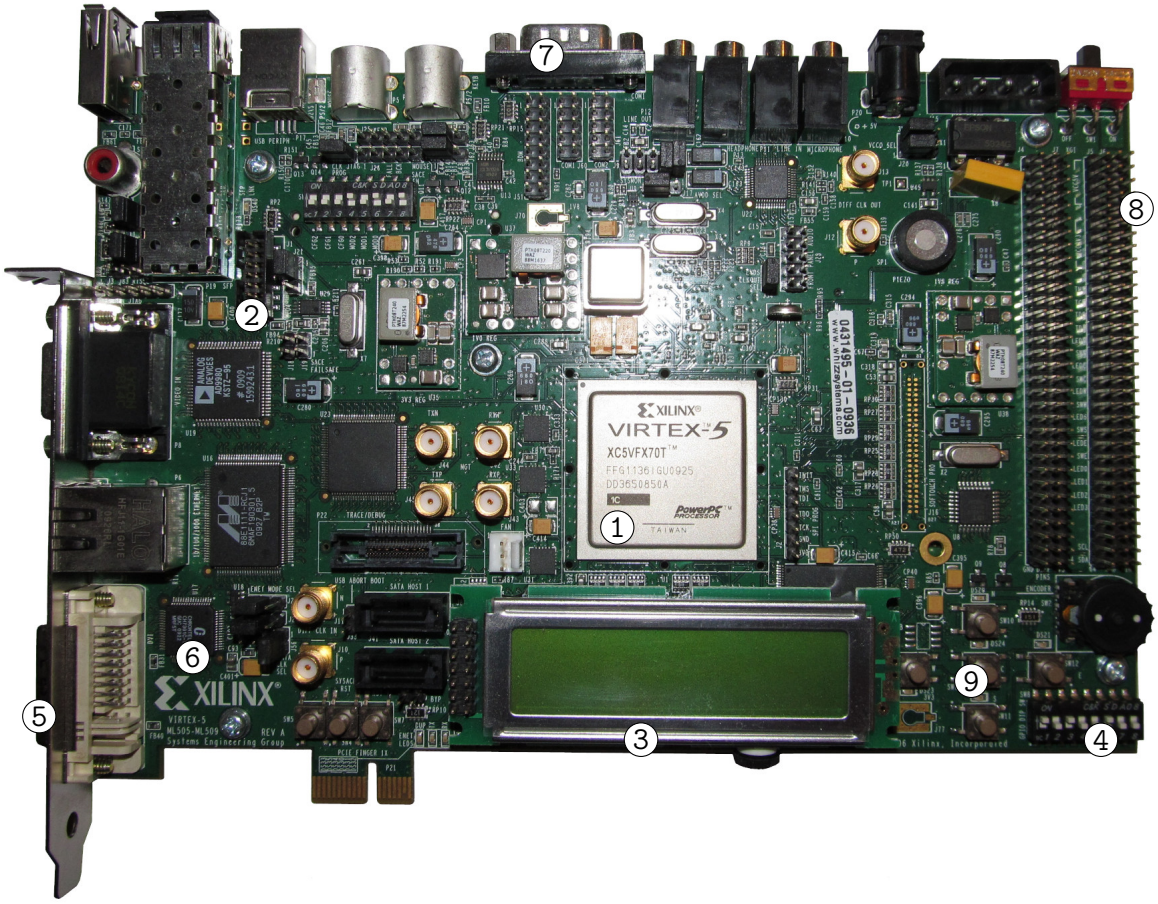

Abbildung 2.1: ML507-Entwicklungsboard mit 1: FPGA, 2: JTAG-Anschluss, 3: LEDs (unter dem Display), 4: DIP-Schalter, 5: DVI-Anschluss, 6: Chrontel CH7301C zur DVI-Signalerzeugung, 7: UART-Anschluss, 8: GPIO-Pins für SPI und 9: Reset-Schalter. Der DDR2-Speicher und die CompactFlash-Karte befinden sich auf der Rückseite.

digitalen Übertragung von Videosignalen [14]. Es ist abwärtskompatibel zum analogen VGA-Anschluss, indem der DVI-Stecker optional auch Pins für ein analoges Signal enthält, was einfache passive Adapter ermöglicht. Eine Grafikkarte muss jedoch kein analoges Signal erzeugen und darf auch ausschließlich digitale Signale ausgeben. Inzwischen wurde DVI bei neuen Geräten größtenteils von dem kompatiblen HDMI und dem moderneren, paketbasierten DisplayPort abgelöst.

DVI nutzt dasselbe grundlegende Signalformat wie VGA, das wiederum auf den ursprünglich für Anzeigegeräte mit Kathodenstrahlröhren (CRT) entwickelten Standards für das analoge Fernsehen basiert. Dabei werden die Pixel mit einem konstanten Takt

zeilenweise übertragen, wobei ein horizontales Synchronisationssignal (H-Sync) den Beginn einer neuen Zeile markiert. Nach der sequenziellen Übertragung aller Zeilen startet ein vertikales Synchronisationssignal (V-Sync) ein neues Bild. Vor und nach den Sync-Signalen werden jeweils für einige Takte keine Bildinformationen übertragen (Front- und Back-Porch, zusammen mit den Sync-Signalen auch als Austastlücke bezeichnet). Die horizontalen Porches dienen bei älteren CRT-Geräten dazu, einem per AC-Kopplung angebundenen Signal genügend Zeit zur Stabilisierung auf Null zu geben, damit der Elektronenstrahl beim Sprung auf den Anfang der nächsten Zeile nicht sichtbar ist [16]. Während der horizontalen Back-Porch wurde außerdem bei Farbfernsehsystemen ein weiteres, hochfrequentes Synchronisationssignal übertragen, mit dem die auf das monochrome Signal modulierten Farbinformationen dekodiert werden konnten. Die vertikale Austastlücke wurde dagegen oft für die Übertragung zusätzlicher Informationen wie z. B. Untertitel oder Teletext verwendet. Bei digitalen Monitoren sind theoretisch keine Porches nötig, jedoch werden diese aus Kompatibilitätsgründen weiterhin unterstützt.

Bei „Single-Link-DVI“ werden die drei Farben eines Pixels parallel über drei differenzielle Signalkanäle übertragen. Dabei kommt das „Transition-Minimized Differential Signaling“ (TMDS) zum Einsatz, das die jeweils 8 b an Farbinformation oder während der Austastlücke zwei Steuerungssignale als 10 b-Worte codiert. Das codierte Wort wird anschließend seriell übertragen. TMDS sorgt dafür, dass im Mittel gleich viele Einsen wie Nullen bei einer möglichst geringen Anzahl an Signalwechseln übertragen werden. H- und V-Sync werden als Steuersignale des blauen Farbkanals übertragen und dienen aufgrund ihrer eindeutigen TMDS-Codierung gleichzeitig als Synchronisierungssignal für den Empfänger. „Dual-Link-DVI“ besitzt zur Erhöhung der Bandbreite drei weitere Kanäle, womit bei höheren Auflösungen zwei Pixel parallel übertragen werden.

2.2 DDR2-SDRAM

Als Arbeitsspeicher ist auf dem Entwicklungsboard ein 256 MiB großes DDR2-Speichermodul vorhanden. Dabei bezeichnet Double Data Rate 2 Synchronous Dynamic Random Access Memory (DDR2-SDRAM) einen Standard für die synchrone Anbindung von flüchtigem Speicher mit wahlfreiem Zugriff [17], wobei die Daten mit doppelter Datenrate übertragen werden. Das bedeutet, dass pro Taktzyklus auf einer Leitung 2 b übertragen werden, eines bei der steigenden Flanke und das andere bei der fallen-

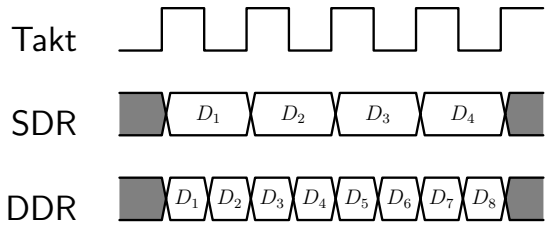

Abbildung 2.2: Bei doppelter Datenrate (DDR) werden in der gleichen Zeit doppelt so viele Daten übertragen wie bei einfacher Datenrate (SDR).

den Flanke des Taktes (siehe Abbildung 2.2). Die Breite des Datenbusses beträgt immer 64 b, wobei dazu typischerweise mehrere Speicherchips auf einem Speichermodul parallel betrieben werden. Innerhalb eines Speicherchips befinden sich mehrere Bänke, welche als Array mit Zeilen und Spalten aufgebaut sind. Die Adresse setzt sich aus dem Index der Bank, Zeile und Spalte zusammen. Es existieren auch Module mit mehreren sog. Rängen, wobei es sich logisch gesehen um die Kombination mehrerer Module auf einem physischen Speichermodul handelt, die sich den Daten- und Adress-Bus teilen. Beim Zugriff wird immer genau ein Modul mit Hilfe eines ChipSelect-Signales aktiviert, welches in diesem Fall ebenfalls ein Teil der Adresse ist.

Während die Adressen und Kommandos synchron über einen gemeinsamen Takt übertragen werden, besitzen immer 8 b des Datenbusses ein eigenes sog. Abtastsignal (engl. strobe), was im Prinzip einem Taktsignal entspricht. Damit lässt sich durch parallele Verlegung der Signale einfach sicherstellen, dass die Verzögerungen bei den acht Signalen und dem zugehörigen Abtastsignal gleich sind, auch wenn verschiedene Abtastsignale unterschiedliche Verzögerungen aufweisen. Bei jeder Lese- oder Schreiboperation werden immer vier oder acht Worte direkt nacheinander übertragen (Burst), was aufgrund der Wortbreite von 64 b insgesamt 32 B oder 64 B entspricht. Diese Burst Length (BL) wird bei der Initialisierung des Speichermoduls durch den Speichercontroller festgelegt.

2.3 FPGAs

Der RISC-V-Prozessor der Workstation wird auf dem Virtex-5-FPGA implementiert, das den Hauptbestandteil des Entwicklungsboards darstellt. Ein Field Programmable Gate Array (FPGA) ist ein programmierbarer integrierter Schaltkreis, der beliebige,

kombinatorische oder sequenzielle Schaltungen nachbildet. Dazu besitzt das FPGA in einem Array angeordnete Slices, die jeweils eine oder mehrere Lookup-Tabellen (LUT) kombiniert mit einem Flipflop enthalten. Jede LUT simuliert typischerweise eine beliebige vier- bis sechsstellige logische Funktion. Dazu wird die komplette Wertetabelle der Funktion in statischen RAM-Zellen gespeichert und die Funktionsparameter wie ein Index zur Auswahl des aktuellen Wertes verwendet. Dieser wird anschließend entweder in dem Flipflop gespeichert oder direkt an den Ausgang des Slices weitergegeben. Durch die Kombination mehrerer Slices über ein ebenfalls konfigurierbares Verbindungsnetz zwischen den Slices lassen sich so beliebig komplexe logische Funktionen nachbilden. Da alle Konfigurationsparameter in statischem RAM gespeichert werden, muss ein FPGA nach jedem Spannungsverlust erneut programmiert werden.

Um häufig verwendete Funktionen wie z. B. Addierer, Multiplexer und Schieberegister nicht mit Hilfe der LUTs implementieren zu müssen, besitzen Slices oft zusätzliche diskrete Schaltungen, die eine deutlich platzsparendere Umsetzung dieser Funktionen ermöglichen. Für Signalverarbeitungszwecke besitzen manche FPGAs außerdem spezielle DSP-Slices, die z. B. schnelle, diskrete Multiplizierer enthalten. Zur Speicherung größerer Datenmengen beinhalten FPGAs darüber hinaus häufig einige größere Blöcke statischen RAMs[1]. Diese werden in einigen vordefinierten Breiten und Tiefen adressiert und besitzen oft zwei unabhängige Schnittstellen, um verschiedenen Teilen einer Schaltung gleichzeitigen Zugriff auf den RAM zu ermöglichen.

2.3.1 Taktdomänen

Eine Taktdomäne bezeichnet eine Menge von logischen Bauteilen, die mit einem gemeinsamen Takt betrieben werden. Ein FPGA besitzt ein spezielles Taktverteilungsnetz, um sicherzustellen, dass alle synchronen Elemente einer Taktdomäne den gewünschten Takt zum selben Zeitpunkt erhalten. Weiterhin unterstützen moderne FPGAs gleichzeitig mehrere verschiedene Taktdomänen. Dies ist z. B. nötig, falls mit mehreren externen Komponenten mit unterschiedlichen Taktraten kommuniziert werden soll.

Um Signale von einer in eine andere Taktdomäne zu übertragen sind spezielle Maßnahmen nötig, da das Eingangssignal eines Flipflops vor und nach der steigenden Flanke des

[1] Bei Xilinx werden diese als BlockRAM bezeichnet.

Takts für eine bestimmte Zeit stabil sein muss (Setup- und Haltezeit). Ein Eingangssignal aus einer anderen Taktdomäne wird sich bei unterschiedlichen Taktraten irgendwann zwangsläufig während der Setup- oder Haltezeit des nächsten Flipflops ändern, was diesen in einen metastabilen Zustand versetzt. Dabei oszilliert der Ausgang eine unbestimmte Zeit lang, bevor er sich schließlich auf einen zufälligen Wert festlegt. Falls dieses Signal direkt weiter verwendet wird, führt dies potenziell zu metastabilen Zuständen bei weiteren Flipflops, was zu falschen Berechnungen und bei CMOS-Technologie zu einem stark erhöhten Stromfluss und damit einem rapiden Temperaturanstieg führt. Um diese Probleme zu vermeiden, wird das Eingangssignal aus der anderen Taktdomäne durch typischerweise drei serielle Flipflops der aktuellen Taktdomäne geleitet. Dadurch hat der erste Flipflop einen kompletten Taktzyklus lang Zeit, sich zu stabilisieren und beeinflusst währenddessen keine weitere Logik. In dem bereits sehr unwahrscheinlichen Fall, dass sich der erste Flipflop nicht rechtzeitig stabilisiert und der zweite in einen metastabilen Zustand gelangt, stabilisiert der dritte Flipflop das Signal. Dadurch ergibt sich eine so geringe Fehlerwahrscheinlichkeit, dass dieses Problem typischerweise während der Lebensdauer eines Produktes nicht auftritt.

Ein zusätzliches Problem tritt auf, falls Daten von einer schnellen in eine langsamere Taktdomäne übertragen werden sollen. In diesem Fall besteht die Möglichkeit, dass der langsame Takt die Daten niemals registriert, falls diese im schnelleren Takt nur für wenige Zyklen verfügbar waren. Eine Lösung stellt hier ein Handshake-Verfahren dar, bei dem beide Seiten der Übertragung so lange warten, bis die jeweils andere Seite bereit ist bzw. gültige Daten vorliegen.

2.3.2 Entwurf einer Schaltung

Die Schaltung für einen FPGA wird entweder als Schaltplan unter Verwendung logischer Gatter wie AND, NOT und Flipflops oder in einer Hardwarebeschreibungssprache wie VHDL oder Verilog entworfen. Dabei wird auf einer abstrakten Ebene die Bewegung der Daten zwischen verschiedenen Registern beschrieben (Registertransferebene). Bei der Synthese wird daraus ebenfalls ein logischer Schaltplan erzeugt. Als nächstes werden die generischen Logik-Gatter des Schaltplans auf die im FPGA vorhandenen Primitive wie LUTs und Flipflops abgebildet. Diese werden anschließend konkreten Slices zugewiesen und korrekt verbunden. Schließlich wird eine Konfigurationsdatei erzeugt, mit welcher der FPGA programmiert wird.

Bei getakteten Schaltungen wird nach der Platzierungs- und Verbindungsphase außerdem durch eine Timing-Analyse überprüft, ob bei allen Flipflops die Setup- und Haltezeiten eingehalten wurden und die Laufzeiten zwischen den Registern nicht zu groß sind. Dazu werden in einer sog. Constraint-Datei die Taktfrequenz oder alternativ auch maximale Verzögerungen einzelner Verbindungen angegeben. In dieser Datei wird außerdem festgelegt, welche Signale der nachgebildeten Schaltung mit den IO-Pins des FPGA zur Kommunikation mit externen Komponenten verbunden werden.

2.4 Chisel

Bei Chisel (Constructing Hardware In a Scala Embedded Language) handelt es sich um eine freie Hardwarebeschreibungssprache, die als eingebettete, domänenspezifische Sprache (embedded DSL) in Scala implementiert ist. Entwickelt wurde Chisel an der Universität von Kalifornien, Berkeley (UCB) als Alternative zu Verilog und VHDL [3]. So ist die Semantik der Sprache, anders als bei Verilog und VHDL, nicht primär auf die Simulation, sondern auf die Synthese zugeschnitten. Das führt insbesondere dazu, dass beim Entwurf nicht darauf geachtet werden muss, ob ein bestimmtes Sprachkonstrukt (effizient) synthetisierbar ist.

Darüber hinaus eignet sich Scala als objektorientierte und funktionale Sprache mit einem generischen Typsystem und einer Laufzeitumgebung mit Unterstützung für Reflexion gut für die Metaprogrammierung. So müssen die Bit-Breiten von Signalen nicht explizit angegeben werden, sondern werden aus den Schnittstellen inferiert. Außerdem ermöglichen sog. Bundles die Zusammenfassung mehrerer Signale, um diese wie ein einziges Signal mit einem Befehl zu verbinden. Weiterhin dürfen Signale als Parameter an Funktionen übergeben und neue Signale zurückgegeben werden. Innerhalb von Funktionen dürfen dabei weitere Module instanziiert werden. Dies ermöglicht die Definition einer kompletten Hierarchie aus korrekt verbundenen Modulen über einen einzigen Befehl aus geschachtelten Funktionsaufrufen. So wird in folgendem Beispiel ein Register `data` definiert, das bei jeder fünften Datenübertragung im Kanal aktualisiert wird:

```
val data = channel.data holdUnless Counter(channel.fire(), 5)._2
```

Diese Abstraktionen ermöglichen Chip-Generatoren wie Rocket Chip [1], die aus einer deklarativen Konfiguration (z. B. Anzahl Prozessorkerne, Größe und Assoziativität der

Caches, zusätzliche Interfaces und Peripheriegeräte, etc.) automatisch einen kompletten System-on-a-Chip (SoC) erzeugen. Ein solcher von Rocket Chip generierter SoC bildet auch die Grundlage der Workstation und wird im nächsten Kapitel vorgestellt.

Bei der Synthese eines mit der Chisel-Bibliothek erstellten Scala-Programms wird die Hardware nicht direkt aus dem Scala-Code generiert, sondern das Ausführen des Programms erzeugt eine Hardwarebeschreibung auf Registertransfer-Ebene im Firrtl-Format (Flexible Internal Representation for RTL). Diese wird wiederum von einem Firrtl-Compiler weiterverarbeitet und schließlich zu Verilog kompiliert, das sich sowohl für die Synthese auf einem FPGA oder die Erstellung eines ASICs, als auch für die Simulation mit Simulationswerkzeugen wie z. B. Verilator eignet. Für die Einbindung eines vorhandenen Verilog- oder VHDL-Moduls wird in Chisel eine sog. BlackBox definiert.

2.5 RISC-V

RISC-V ist eine freie, ursprünglich an der UCB entwickelte und von der RISC-V Foundation spezifizierte Befehlssatzarchitektur (engl. Instruction Set Architecture, ISA). Sie ist unter der Creative Commons BY 4.0-Lizenz lizenziert, die sowohl proprietäre als auch offene Implementierungen in Prozessoren ermöglicht. Das Ziel ist eine weite Abdeckung vieler Anwendungsfälle von Forschung und Lehre über eingebettete Mikroprozessoren bis hin zu High Performance Computing. Dadurch erfährt die ISA bereits eine breite Unterstützung an Universitäten und bei kommerziellen Anbietern. So gehören neben der von den ursprünglichen RISC-V-Entwicklern gegründeten Firma SiFive u. a. Google, IBM, Nvidia, Western Digital und Qualcomm zu den Gründungsmitgliedern der Foundation.

2.5.1 Architektur

RISC-V unterstützt 32 b und 64 b als Wortbreiten und ist für eine zukünftige Erweiterung auf 128 b vorbereitet. Die User-Level ISA [12] ist modular aufgebaut. So gibt es eine Basis-ISA (I) mit Unterstützung für einfache arithmetische und logische Operationen, bedingte und unbedingte Sprünge sowie Speicherzugriffe. Daneben existieren optionale Erweiterungen für Hardware-Multiplikation und -Division (M), atomare Speicheroperationen (A) und Unterstützung für IEEE 754-Gleitkommaoperationen in verschiedenen

Breiten (F, D und Q). Die von einem Prozessor unterstützten Erweiterungen werden zusammen mit der Wortbreite als RV64IMAFD angegeben. Die Kombination IMAFD wird dabei auch mit G abgekürzt.

Die ISA verwendet 32 Register, wobei das erste alle geschriebenen Werte ignoriert und beim Lesen immer 0 zurückliefert, was die Einsparung einiger redundanter Befehle ermöglicht. So werden durch geschickte Nutzung des 0-Registers nur 2 Befehle benötigt, um sämtliche Arten von unbedingten Sprüngen, Funktionsaufrufen und Funktionsrückgaben zu realisieren. Allgemein liegt der Fokus auf einer geringen Anzahl Befehle, die von Prozessoren möglichst einfach decodierbar sein sollen. So befinden sich z. B. die Quell- und Zielregister innerhalb der Instruktionen immer an der selben Stelle. Für die Programmablaufsteuerung existieren außerdem ausschließlich einige bedingte Sprünge, die zwei Register auf Gleichheit, Ungleicheit oder bezüglich der Größe vergleichen.

Um Programme trotz der reduzierten ISA kompakt zu halten, spezifiziert RISC-V eine Erweiterung zur Kompression der Befehle (C). Dabei werden für die am häufigsten verwendeten und normalerweise 32 b langen Befehle alternative 16 b-Codierungen definiert. Diese dürfen beliebig mit den unkomprimierten Befehlen kombiniert werden, da die Codierungen so gewählt sind, dass anhand der ersten Bits erkennbar ist, ob es sich um einen komprimierten Befehl handelt. Komprimierte Befehle reduzieren neben der Größe des Programms auch die Auslastung des Befehlscaches, wodurch gleichzeitig mehr Befehle im Cache vorgehalten werden und die Performance erhöht wird. Außerdem spezifiziert die ISA, dass bedingte Rückwärtssprünge üblicherweise und bedingte Vorwärtssprünge normalerweise nicht ausgeführt werden. Dies ermöglicht es einem optimierenden Compiler, den Code so anzuordnen, dass die Sprungvorhersage eines Prozessors selbst einen erstmals ausgeführten Sprung üblicherweise korrekt vorhersagt.

Neben der User-Level ISA beschreibt die zum Zeitpunkt dieser Arbeit nur als Entwurf vorliegende Privileged Architecture-Spezifikation [13] die Schnittstelle für Betriebssysteme. Diese besteht aus Statusregistern für die Konfiguration von Interrupts und Exceptions und privilegierten Befehlen, welche die Implementierung von Interruptroutinen und Systemaufrufen erlauben. Dazu werden zusätzlich zum User-Level der Machine-Level für Initialisierungszwecke nach dem Reset und der Supervisor-Level für Betriebssysteme eingeführt. Weiterhin werden virtuelle Speichersysteme für RV32 und RV64 beschrieben, wobei für RV64 39 b und 48 b breite virtuelle Speicheradressen unterstützt werden.

2.5.2 Auswahl eines Cores

Im Verlauf dieser Arbeit wurden verschiedene freie RISC-V-Prozessoren (Cores) genauer betrachtet, um eine Basis für die Implementierung der Workstation zu finden.

Zu Beginn wurde River [18] getestet, da dieser in VHDL geschrieben ist, zumindest teilweise die Privileged Architecture implementiert und bereits ein Xilinx-ISE-Projekt für ein Virtex-6-Board mitliefert. Während das Projekt relativ problemlos für das ML507-Board adaptiert werden konnte, traten bei der Synthese für Virtex-5 eine Reihe von Fehlern auf, die zum damaligen Zeitpunkt nicht behoben werden konnten. Da außerdem eine Unterstützung von Linux aufgrund der unvollständigen Implementierung der Privileged Architecture (u. a. keine Unterstützung für virtuellen Speicher) entweder eine Implementierung der fehlenden Teile oder eine Anpassung von Linux erfordert hätte, wurde die Verwendung dieses Cores nicht weiter verfolgt.

Als nächstes wurde VexRiscv getestet. Dabei handelt es sich um einen vom Entwickler als „FPGA-freundlich“ [4] bezeichneten, in SpinalHDL geschriebenen, RV32-Core. Der Prozessor ist dabei modular aufgebaut und lässt sich somit leicht durch Plugins erweitern. Bei SpinalHDL handelt es sich, wie bei Chisel, um eine als Scala-DSL implementierte Hardwarebeschreibungssprache. Der Core konnte innerhalb kürzester Zeit erfolgreich für das ML507-Board synthetisiert und einige Beispielprogramme getestet werden.

Parallel dazu wurde die auf dem Rocket-Prozessor basierende Freedom U500 Plattform genauer betrachtet. Während die Anpassung an das ML507-Board aufgrund der Komplexität deutlich länger dauerte, sorgte die Konfigurierbarkeit von Rocket dafür, dass der FPGA optimal ausgenutzt werden konnte (siehe dazu Abschnitt 3.3). Da es sich bei Rocket außerdem um einen 64 b-Chip handelt und Linux offiziell unterstützt wird, wurde schließlich Rocket als Basis für die Implementierung dieser Arbeit ausgewählt.

3 Rocket Chip Generator

Rocket Chip ist eine Sammlung von Chisel-Hardwaregeneratoren zur Erzeugung von RISC-V-SoCs [1]. Das Projekt ist dabei sowohl unter der Apache 2- als auch der 3-Klausel-BSD-Lizenz lizenziert, die beide mit der GPLv3 kompatibel sind. Als Basis dient der in-order Skalarprozessor Rocket. Dies bedeutet, dass der Prozessor pro Takt einen Befehl in der vom Programm vorgegebenen Reihenfolge ausführt. Rocket unterstützt RV32GC und RV64GC inklusive Privileged Architecture, virtuellen Speicher mit hardwarebeschleunigter Adressauflösung und besitzt eine konfigurierbare Sprungvorhersage und anpassbare Daten- und Befehls-Caches. Außerdem bietet er eine Schnittstelle zur einfachen Anbindung von Co-Prozessoren wie z. B. eine Vektoreinheit, die mehrere Datenworte gleichzeitig verarbeitet.

Der Prozessor ist modular aufgebaut, so dass viele der Komponenten wie die Gleitkommaeinheit (FPU), die Caches oder die Register für die Privileged Architecture wiederverwendbar sind. Dies nutzt z. B. der superskalare Berkeley Out-of-Order Prozessor (BOOM, [19]) mit Unterstützung für spekulative Befehlsausführung. BOOM implementiert die selbe Schnittstelle wie Rocket, so dass für die Erzeugung eines SoCs ebenfalls Rocket Chip verwendet wird. Rocket Chip ist dabei sogar in der Lage, ein SoC mit mehreren Rocket- und BOOM-Prozessoren und einem gemeinsamen Speichersystem zu erstellen.

Zur Anbindung von Arbeitsspeicher und Peripheriegeräten wie seriellen Schnittstellen oder PCIe-Bussen wird das TileLink-System verwendet. Die Kontroll- und Datenregister der Peripheriegeräte werden dabei in den gleichen Adressraum wie der Arbeitsspeicher abgebildet (engl. Memory Mapped I/O, MMIO), so dass mit normalen LOAD/STORE-Befehlen darauf zugegriffen wird. Der folgende Abschnitt bietet einen kurzen Überblick über TileLink und die darauf aufbauende Diplomacy-Bibliothek.

3.1 TileLink und Diplomacy

TileLink (TL) ist ein offener Standard für ein Bussystem zur kohärenten Anbindung von Speicher und MMIO-Geräten (Slaves) an einen oder mehrere Prozessoren (Master) innerhalb eines SoCs [25]. Es wurde für RISC-V entwickelt und erstmals in Rocket Chip implementiert. Ein TL-Netzwerk ist ein azyklischer, gerichteter Graph aus Agenten, wobei ein Agent die Rolle eines Masters, Slaves oder, z. B. im Falle eines Caches, auch beide Rollen einnimmt. Eine Verbindung besteht immer zwischen einem Master und einem Slave und enthält mehrere Kanäle, die jeweils in einer Richtung Nachrichten übertragen und den unterschiedlichen Prioritäten der Nachrichten entsprechen. Auf Kanal A werden Lese- und Schreibanfragen des Masters an den Slave übertragen. Diese werden anschließend auf Kanal D vom Slave beantwortet. Dabei werden neben den Daten auch eine ID zur Identifikation des Masters und die Adresse und Größe der zu schreibenden oder lesenden Daten übertragen. Falls diese größer sind als die Breite des Datenbusses des Kanals, werden die Daten in mehreren Takten übertragen.

Agenten, die Daten zwischenspeichern wollen, müssen drei weitere Kanäle (B, C und E) unterstützten. Zusammen mit den bereits vorhandenen Kanälen bilden diese die Grundlage für die Implementierung eines Cache-Kohärenz-Protokolls, indem ein Master vor dem exklusiven Cachen der Daten die dafür nötigen Rechte vom zugehörigen Slave erlangen muss. Exklusive Rechte sind nötig, wenn die Daten verändert werden sollen. Wenn nun ein anderer Master auf die Daten zugreifen möchte, muss der ursprüngliche Master die Daten zuerst freigeben und dabei eventuelle Änderungen an den Slave zurückschreiben, bevor die Daten an den neuen Master weitergegeben werden.

Alle Kanäle verwenden ready- und valid-Signale für die Koordinierung der Übertragung. Der Sender signalisiert dabei mit einem aktiven valid-Signal, dass die anliegenden Daten gültig sind und übertragen werden sollen. Der Empfänger signalisiert mit einem aktiven ready-Signal, dass er bereit ist, eine Nachricht anzunehmen. Eine Nachricht wird genau dann übertragen, wenn in einem Takt beide Signal aktiv sind. Um Deadlocks des TL-Netzwerkes zu vermeiden, müssen dabei bestimmte Regeln eingehalten werden. Grundsätzlich müssen Nachrichten mit einer höheren Priorität zuerst verarbeitet werden, wobei Kanal A die niedrigste und Kanal D die höchste Priorität besitzt. Außerdem darf ein Empfänger eine Anfrage nur ablehnen, wenn er gerade selbst versucht, auf einem Kanal mit höherer Priorität eine Antwort an den Sender zu übermitteln. Weiterhin darf ein Empfänger eine bestimmte Zeitspanne lang gar keine Nachrichten annehmen, so

lange die Zeitspanne eine feste Obergrenze besitzt und zwischen mehreren dieser Pausen mindestens eine Nachricht akzeptiert wird. Darüber hinaus muss ein Sender eine kombinatorische Antwort akzeptieren, d. h. eine Antwort, die im selben Takt wie die zugehörige Anfrage übermittelt wird.

Um das korrekte Verbinden aller Agenten in einem TL-Netzwerk zu vereinfachen, wurde für Rocket Chip das Diplomacy-Framework [7] entwickelt. Dabei handelt es sich um eine auf Chisel aufbauende Bibliothek, die automatisch Parameter in einem Busnetzwerk wie TL überprüft und aushandelt. Dazu gibt jeder Agent des TL-Netzwerkes u. a. an, welche Nachrichten er unterstützt, wie groß die zu übertragenen Daten minimal und maximal sein dürfen, ob Daten gecached werden und welche Adressbereiche unterstützt werden. Anhand dieser Parameter überprüft Diplomacy in einer ersten Phase, ob die angegebenen Verbindungen zwischen den Agenten korrekt sind, und handelt gültige Werte für nicht spezifizierte Parameter wie die Breite der Datenverbindungen der einzelnen Kanäle aus. Außerdem wird eine Liste der im System vorhandenen Adressbereiche erstellt, wobei dort festgelegt wird, ob eine Adresse gelesen, beschrieben oder ausgeführt werden darf. In einer zweiten Phase erhalten die Agenten Zugriff auf die ausgehandelten Parameter und generieren exakt die dafür nötigen Schaltungen. Sie lassen dabei Signale oder Kanäle aus, die in einer Verbindung nicht verwendet werden.

Diplomacy funktioniert dabei nicht nur mit TL, sondern auch mit anderen Bussystemen wie AXI und sogar, wenn verschiedene Bussysteme gemeinsam verwendet werden. Die dazu nötigen Adapter-Komponenten konvertieren die in den Parametern kodierten Anforderungen des einen Bussystems jeweils in Parameter des anderen Bussystems. In Rocket Chip sind solche Adapter für die Bussysteme AXI4, APB und AHB implementiert.

3.2 Freedom Plattform

SiFive vertreibt aktuell zwei verschiedene, auf Rocket Chip basierende RISC-V-Produktlinien, die Freedom Everywhere und die Freedom Unleashed Plattform [23]. Die Everywhere Plattform ist für eingebettete Systeme und Internet of Things-Geräte gedacht und besitzt mit dem E310 aus der E300-Reihe einen RV32-SoC ohne FPU und virtuelle Speicherverwaltung. Der U540-SoC aus der U500-Reihe mit vier RV64-Kernen unterstützt als Teil der Unleashed Plattform dagegen sowohl Gleitkommaoperationen

als auch virtuellen Speicher und ist damit in der Lage, ein vollständiges Linux-System auszuführen. Der Chip ist aktuell nur als Teil des HiFive Unleashed-Entwicklungsboards erhältlich.

Beide SoCs sind auch als freie, FPGA-basierte Entwicklungsplattformen unter Apache 2-Lizenz verfügbar [22]. Bei der U500-Plattform fehlen hier zwar die auf dem Unleashed-Board verfügbaren, proprietären Schnittstellen wie Ethernet oder DDR4-RAM, alle von SiFive selbst entwickelten Hardwaremodule sind jedoch ebenfalls frei verfügbar. Dazu gehören vor allem grundlegende Schnittstellen wie UART, SPI, I²C, JTAG und ein PWM-Controller. Die Freedom-Plattformen sind darüber hinaus „untethered“. Dies bedeutet, dass sie in der Lage sind, auf dem FPGA ohne einen zusätzlich angeschlossenen, externen Computer selbstständig zu booten. Dazu enthält das SoC einen 8 kiB großes BootROM mit einem darauf enthaltenen Bootloader, der Bootimages über SPI von einer SD-Karte lädt.

3.3 Anpassung an ML507

Um als Basis für die RISC-V-Workstation zu dienen, musste Rocket an das ML507-Entwicklungsboard angepasst werden. Als Basis für diese Anpassung diente die Implementierung der U500-Plattform für das VC707-Board. Dabei handelt es sich um eine Evaluationsplattform für das Virtex-7-FPGA XC7VX485T. Da dieses im Vergleich zu dem auf dem ML507 verbauten Virtex-5 XC5VFX70T fast siebenfach größer ist (76k statt 11k Slices), musste die Konfiguration deutlich verkleinert werden. So wird statt 4 Kernen nur 1 Kern verwendet, wodurch auch ein Großteil der für die Cache-Kohärenz eines Mehrkernsystems nötigen Logik entfällt. Außerdem wurden die Caches deutlich verkleinert (je 4 kiB statt 32 kiB Instruktions- und Daten-Cache) und eine langsamere Hardwaremultiplikationseinheit aktiviert, so dass im Ergebnis ein vollständiger Rocket-Kern inklusive virtueller Speicherverwaltung, IEEE 754-Gleitkommaeinheit, DDR2-Interface und Terminal mit DVI-Signalerzeugung auf das FPGA passt und mit einem Takt von 60 MHz ausgeführt wird.

Die Hardwareausstattung der beiden Evaluationsplattformen ist im Gegensatz dazu ziemlich ähnlich. So besitzen beide neben den üblichen GPIO-Pins, LEDs und Tastern eine Schnittstelle für DDR-Speicher[1], Flash-Speicher für FPGA-Konfigurationen,

[1] DDR2 auf dem Ml507 und DDR3 auf dem VC707

Videoausgänge und weitere Schnittstellen wie PCIe und Ethernet, die in dieser Arbeit nicht verwendet wurden.

Für den Speicherzugriff beinhalten Xilinx ISE (bis Virtex-7) bzw. Vivado (ab Virtex-7) den sogenannten Memory Interface Generator (MIG), der einen quelloffenen, aber proprietären IP-Kern für den Zugriff auf DDR-Speicher generiert. Ab Virtex-6 besitzt dieser Kern ein AXI4-Interface, ein von ARM entwickeltes und heute weit verbreitetes Busprotokoll für die Verbindung verschiedener Komponenten innerhalb eines Chips. RocketChip beinhaltet verschiedene Konverter von TileLink zu AXI4 und umgekehrt und verwendet diese u. a. für die Anbindung des Arbeitsspeichers und des PCIe-Bussystems. Da der MIG-Kern unter Virtex-5 jedoch nur eine proprietäre und nicht-standardisierte Schnittstelle besitzt, musste für das ML507-Board ein neuer TileLink-Wrapper für den MIG-Kern entwickelt werden, was in Kapitel 4.1 genauer beschrieben ist.

Die Situation bei der PCIe-Anbindung ist ähnlich. So unterstützt die LogiCORE AXI/PCIe-Bridge [27] nur Virtex-6-FPGAs oder neuer. Da eine PCIe-Anbindung des Prozessors für das primäre Ziel der Arbeit nicht genügend relevant ist und eine Implementierung zusätzlichen Platz auf dem FPGA belegen würde, wurde das Rocket-PCIe-Interface für die Implementierung auf dem ML507-Board deaktiviert.

Da nur das bereits abgekündigte Xilinx ISE Framework Unterstützung für Virtex-5-FPGAs bietet, war der Autor auf den VHDL-Standard von 1993 beschränkt. Jedoch wurde eine von ISE optional angebotene, unvollständige Vorschau auf VHDL 2002 aktiviert. Verilog wird in Version 2001 unterstützt, was den von Chisel bzw. dem Firrtl-Compiler erzeugten Code problemlos unterstützt. Nur bei direkt in Verilog geschriebenen Modulen waren einige kleine Änderungen nötig. So unterstützt z. B. die readmemh-Funktion, die für die Initialisierung des Boot-ROM aus einer Textdatei verwendet wird, unter ISE nur statische Pfadangaben. Außerdem wurde auch der normalerweise nur für Virtex-6-FPGAs und neuer verwendete Parser der ISE per Kommandozeilenparameter aktiviert [10], da der alte Parser BlockRAMs bei unvollständiger Initialisierung (der Bootloader wird direkt aus seinem C-Quellcode kompiliert und füllt deshalb nie exakt das Boot-ROM aus) nicht initialisiert.

4 Peripherie

4.1 Speichercontroller

Als Hauptspeicher für den Prozessor wird der auf dem Entwicklungsboard verfügbare DDR2-RAM verwendet. Dabei handelt es sich um das 256 MiB große Speichermodul MT4HTF3264HY von Micron im SODIMM-Format, das mit den vollen 64 b Datenbreite an das FPGA angeschlossen ist. Als Speichercontroller dient ein von Xilinx MIG erzeugter IP-Kern, der vom `memory_controller`-Modul gesteuert wird. Dieses ist wiederum über einen in Chisel geschriebenen TL-Slave an den Speicherbus des Prozessors angeschlossen.

4.1.1 Xilinx MIG

Beim Memory Interface Generator (MIG) handelt es sich um einen in Xilinx ISE integrierten Generator für Speichercontroller für FPGAs. Bei Virtex-5-FPGAs werden DDR- und DDR2-SDRAM mit bis zu 267 MHz unterstützt. Um die Takterzeugung zu vereinfachen (das ML507-Board bietet bereits einen 200 MHz Takt) und den Platzbedarf zu reduzieren, wurde der Speicher nur mit 200 MHz angebunden. Dies entspricht der langsamsten DDR2-Variante DDR2-400 und bietet eine maximale theoretische Übertragungsrate von 3,2 Gb/s. Als Burst Length wird 4 (256 b) verwendet, da die Daten auf dem FPGA parallel weiterverarbeitet werden und es bei einer BL von 8 mit den dann 512 b breiten Datenbussen zu Routingproblemen kommt. Der generierte Code darf nur an andere Xilinx-Kunden mit gültiger Lizenz weitergegeben werden. Ein daraus kompilierter Bitstream darf zwar grundsätzlich weitergegeben werden, allerdings nur zur Verwendung auf Xilinx-Produkten. Beides ist inkompatibel mit freien Open Source-Lizenzen, weshalb der Code nicht im finalen Projekt enthalten ist und somit vor der Verwendung mit MIG generiert werden muss.

Der erzeugte Speichercontroller ist ein generisches VHDL-Modul, das mit den genauen Timings und dem Aufbau des Speichermoduls parametrisiert wird. Damit funktioniert der synthetisierte Controller nicht mit anderen Speichermodulen. MIG bietet einen Assistenten, mit dem diese Parameter konfiguriert werden. Dort sind auch eine Reihe vorkonfigurierter Profile für verschiedene auf Xilinx Entwicklungsboards enthaltene Speichermodule auswählbar, u. a. für das auf dem ML507 enthaltene Micron MT4HTF3264HY. Neben den Modulparametern werden dort auch die IO-Pins konfiguriert, an die der Speicher angeschlossen ist. Leider steht dabei kein Entwicklungsboard wie das ML507 mit bereits fest vorgegebenen Pins zur Auswahl. Zwar gibt es die Möglichkeit, eine vorhandene Constraint-Datei zu laden, allerdings werden dabei einige wenige Pins nicht korrekt erkannt und lassen sich auch nicht manuell auswählen. Da sich der Assistent nur fortsetzen lässt, wenn alle Pins korrekt ausgewählt wurden, kann dieser Teil des Assistenten nicht sinnvoll verwendet werden. Um den Assistenten dennoch erfolgreich abzuschließen, wird ein komplett neues Pin-Layout erzeugt und die generierte Constraint-Datei anschließend nicht verwendet. Diese muss stattdessen von Hand erstellt werden.

Die selbst erstellte Constraint-Datei besteht aus drei Teilen. Der erste Teil sind die Zuweisungen der DDR2-Signale an die korrekten IO-Pins. Diese werden direkt aus der Master-Constraint-Datei für das ML507-Board [30] übernommen. Der zweite Teil sind die vom konfigurierten Takt abhängigen Timing-Spezifikationen. Diese sind unabhängig von den Pins und können somit aus der von MIG erzeugten Constraint-Datei übernommen werden. Der letzte Teil ist die Zuweisung passender IDELAY-Elemente an die acht DQS-Signale. IDELAY-Elemente sind Puffer mit einer einstellbaren Signalverzögerung und werden für das korrekte Einlesen der vom Speichermodul gelesenen Daten verwendet [28, S. 373]. Zu diesem Zweck müssen diese möglichst nahe an den zugehörigen IO-Pins platziert werden. Die dazu nötigen Anpassungen sind in einem Hilfebeitrag von Xilinx beschrieben [26], wobei das dort erwähnte Skript nicht mehr verfügbar ist und alle Anpassungen von Hand erfolgen müssen.

Das Interface eines MIG-Speichercontrollers besteht unter Virtex-5, neben den Takt- und DDR2-Signalen, primär aus den Schnittstellen der eingebauten Warteschlangen für Adressen und zu schreibenden Daten. Die zu lesenden Daten werden dagegen nicht in einer Warteschlange zwischengespeichert und müssen direkt weiterverarbeitet werden. Die Datenbreite für Lese- und Schreibports entspricht der doppelten DDR2-Datenbreite (128 b), was es ermöglicht, die Daten für die steigende und fallende Flanke des DDR-

Interfaces zusammen an einer steigenden Flanke zu übertragen. Um einen Schreibvorgang zu starten, müssen die Kontrollsignale entsprechend gesetzt und gleichzeitig die Adresse und die zu schreibenden Daten angelegt werden. Da bei einer BL von 4 immer vier Datenpakete innerhalb von 2 Takten geschrieben werden, muss direkt im darauf folgenden Takt die zweite Hälfte der Daten in die Warteschlange eingefügt werden. Die Adresse bezieht sich dabei auf die Breite des DDR2-Datenbusses (64 b) und muss somit bei sequentiellen Operationen für den nächsten Schreibvorgang um vier erhöht werden. Der Lesevorgang funktioniert analog, nur dass dabei keine zu schreibenden Daten angegeben werden. Die gelesenen Daten werden nach einigen Wartezyklen an zwei direkt aufeinander folgenden Takten ausgegeben und müssen unverzüglich weiterverarbeitet oder zwischengespeichert werden. Alle Operationen werden in der Reihenfolge ausgeführt, in der sie in die Warteschlange eingefügt wurden (FIFO).

4.1.2 MIG-Wrapper `memory_controller`

Um die Integration und das Testen des MIG-Moduls zu vereinfachen, wurde vom Autor das VHDL-Modul `memory_controller` implementiert. Es besitzt ein einfacheres Interface, bei dem alle Daten eines Schreib- oder Lesevorgangs in einem einzigen Takt übertragen werden. Dazu wurde die Breite der Schreib- und Leseports abermals auf 256 b verdoppelt. Die erste Hälfte der zu verarbeitenden Daten wird dabei immer direkt an den MIG-Speichercontroller weitergeleitet und nur die zweite Hälfte für den nächsten Takt zwischengespeichert, wodurch keine zusätzlichen Verzögerungen entstehen. Beim Lesen werden die Daten weiterhin nur einen Takt lang ausgegeben, allerdings lassen sie sich nun einfacher in einer Warteschlange zwischenspeichern, da alle Daten auf einmal vorliegen. Außerdem beziehen sich die Adressen nun auf einzelne Bytes und werden automatisch entsprechend konvertiert. Weiterhin ist das Interface komplett generisch und unabhängig von der Schnittstelle des MIG-Speichercontrollers, um potenziell auch die Verwendung anderer Speichercontroller zu ermöglichen, ohne die weitere Integration verändern zu müssen.

4.1.3 TileLink-Integration

Zur Anbindung des Speichers an den Prozessor wurde vom Autor in Chisel ein TL-Slave entwickelt, der das `memory_controller`-Modul mit dem Speicherbus des Rocket-

SoCs verbindet. Da die integrierten Caches immer komplette Cache-Lines (64 B) lesen und schreiben, ist der Speicherbus 512 b breit. Um diesen auf die gewünschten 256 b zu konvertieren wird der in Rocket Chip enthaltene `TLFragmenter` verwendet. Dieser teilt transparent alle zu schreibenden Daten in kleinere Anfragen auf und setzt die Antworten wieder zur ursprünglichen Größe zusammen. Außerdem wird der ebenfalls bereits vorhandene `TLAsyncCrossing`-Puffer verwendet, um die TL-Anfragen und Antworten zwischen den unterschiedlichen Taktdomänen des Prozessors und des Speichers zu transferieren.

Der eigentliche Slave ist im Modul `XilinxML507MIGToTL` implementiert. Es werden nur die grundlegenden Lese- und Schreib-TL-Anfragen unterstützt, da atomare Operationen und Cache-Kohärenz bereits von den Caches weiter oben in der Speicherhierarchie implementiert werden. Auf jede TL-Anfrage muss eine Antwort zurückgeliefert werden, welche die Source-ID und Größe der ursprünglichen Anfrage enthält. Da der MIG-Speichercontroller keine Möglichkeit bietet, bei den Leseanfragen zusätzliche Metadaten mit anzugeben und Schreibanfragen grundsätzlich nicht bestätigt werden, müssen diese Informationen in einer eigenen Warteschlange für ausstehende Antworten zwischengespeichert werden. Aus dieser Warteschlange werden anschließend die TL-Antworten generiert, wobei bei Leseanfragen gewartet werden muss, bis die Daten zur Verfügung stehen, während Schreibanfragen direkt bestätigt werden. Das funktioniert nur, da der MIG-Speichercontroller alle Anfragen in FIFO-Reihenfolge bearbeitet. Damit entspricht bei mehreren Lesevorgängen der erste Eintrag in der Warteschlange der ausstehenden Antworten der ersten an den Speichercontroller übertragenen Anfrage, welche aufgrund der FIFO-Eigenschaft auch als erstes bearbeitet wird und somit als erstes die korrekten Daten zurückliefert. Auch das Bestätigen von eventuell noch nicht ausgeführten Schreibanfragen ist kein Problem, da alle nachfolgenden Anfragen später ausgeführt werden und damit garantiert den von der Schreiboperation aktualisierten Zustand sehen.

Um das Problem der nur in einem Takt verfügbaren gelesenen Daten zu umgehen, werden diese als erstes in eine Warteschlange eingefügt. Diese muss mindestens so lang sein wie die Warteschlange für die ausstehenden Antworten (Ack-Queue), um zu garantieren, dass keine Daten verloren gehen. Da der TL-Slave Anfragen nur annimmt, falls die Ack-Queue nicht bereits voll ist, können nie mehr Leseanfragen gleichzeitig aktiv sein, als Platz in der Warteschlange für die Antworten ist. Erst nachdem eine Leseanfrage beantwortet wurde und dabei auch die entsprechenden Daten aus der Warteschlange entfernt wurden, werden wieder neue Anfragen angenommen.

4.2 Terminal

Als Ausgabemedium für die RISC-V-Workstation dient ein einfaches Text-Terminal mit Unterstützung für die Anzeige von ASCII und einer Implementierung der Steuercodes für die Rücktaste, den Wagenrücklauf und den Zeilenvorschub. Die Auflösung beträgt 640x480 px bei 60 Hz, womit sich inklusive der Austastlücken eine Auflösung von 800x500 px und bei 8 b pro Farbe ein Pixeltakt von 24 MHz ergibt. Das Terminal wurde vom Autor in VHDL implementiert und besteht aus drei Komponenten.

Die erste Komponente ist für die Initialisierung des CH7301C-Chips per I²C[1] verantwortlich, der direkt an die DVI-Buchse angeschlossen ist und die seriellen Pixeldaten erzeugt. Eine zweite Komponente generiert für den CH7301C aus den aktuell im Terminal gespeicherten Zeichen ein paralleles VGA-Signal inklusive Synchronisationssignalen. Die letzte Komponente ist der Zeichenpuffer des Terminals, der die geschriebenen Zeichen und Zeilen in BlockRAMs zwischenspeichert und die Position und Bewegung des Cursors verwaltet.

4.2.1 Initialisierung des CH7301C

Der CH7301C von Chrontel ist für die Codierung des DVI-Singals zuständig. Hierfür ist ein eigener Chip notwendig, da DVI die Pixeldaten seriell überträgt und dadurch vor allem bei höheren Auflösungen schnell Taktraten von mehreren Gigahertz nötig sind, wofür ein FPGA nicht ausgelegt ist. Stattdessen werden die Daten im FPGA mit nur einem Zehntel des eigentlichen Taktes erzeugt und parallel zum CH7301C übertragen. Da für 24 b pro Pixel nur 12 Datenleitungen vorhanden sind, müssen die Daten entweder im DDR-Verfahren oder mit der doppelten Frequenz übertragen werden. Im Falle des Terminals ist die Auflösung so gering, dass sich die Daten problemlos mit dem doppelten Pixeltakt (48 Mhz) übertragen lassen.

Nach dem Reset benötigt der CH7301C einige Mikrosekunden für die Initialisierung und muss anschließend per I²C konfiguriert werden [5]. Dazu wird ein von Digi-Key entwickelter und auf Anfrage für diese Arbeit unter einer freien Lizenz zur Verfügung gestellter IP-Core verwendet, der einen I²C-Master implementiert, welcher das Lesen und Beschreiben der Register in einem I²C-Slave wie dem CH7301C erlaubt.

[1] Ein weit verbreitetes und einfaches serielles Protokoll mit Master-Slave-Architektur

Zuerst müssen die Taktgeneratoren und Ausgänge aktiviert werden. Anschließend wird der Frequenzbereich für verschiedene interne Komponenten eingestellt. Schließlich muss noch festgelegt werden, ob die Daten per DDR oder SDR übertragen werden. Danach wird das VGA-Signal korrekt kodiert und der Monitor sollte sich synchronisieren.

4.2.2 Erzeugung des VGA-Signals

Das vertikale und horizontale Synchronisationssignal wird von einfachen Zählern erzeugt, die mit der Pixelfrequenz der Reihe nach alle Pixel aufzählen. Wenn sich ein Zähler im Synchronisationsbereich (HSync: 656–720 px; VSync: 483–487 px) befindet, wird das entsprechende Synchronisationssignal aktiviert. Gleichzeitig dienen die Zähler als Koordinaten für das Auslesen des aktuellen Zeichens aus dem Zeichenpuffer.

Um ohne aufwändige Division aus den Pixel-Koordinaten die aktuell anzuzeigende Zeile und die Position des Zeichens innerhalb der Zeile zu ermitteln, müssen sowohl die Breite als auch die Höhe eines Zeichens eine Zweierpotenz sein. Bei der gegebenen Auflösung von 640x480 px bietet sich 8x8 px pro Zeichen an, womit sich eine Zeilenlänge von 80 Zeichen und eine Bildschirmhöhe von 60 Zeilen ergibt. Somit entsprechen die vorderen 7 b bzw. 6 b der Pixel-Koordinaten immer dem Index des Zeichens bzw. der Zeile und die hinteren 3 b jeweils der Position innerhalb des 8x8 px großen Zeichens.

Mit diesen Koordinaten wird nun in einem ersten Takt der ASCII-Code aus dem Zeichenpuffer gelesen. Im nächsten Takt wird diesem Code mit Hilfe einer Schriftart eine Glyphe zugeordnet und mit dem hinteren Teil der Koordinaten der aktuelle Pixel innerhalb der Glyphe bestimmt. Falls dieser ein Teil der Glyphe ist, wird Weiß als Pixelfarbe ausgegeben, andernfalls Schwarz. Als Schriftart kommt eine alte, gemeinfreie[2] IBM-Schriftart zum Einsatz, die sequenziell in einem BlockRAM gespeichert wird. Dabei wird der ASCII-Code direkt als Index für das BlockRAM verwendet.

4.2.3 Zeichenpuffer

Im Zeichenpuffer werden die geschriebenen Zeichen zeilenweise zwischengespeichert, wobei, wenn nötig, die ältesten Daten wie in einem Ringpuffer überschrieben werden. Dazu

[2]Gemeinfreiheit hier im angelsächsischen Sinn: Public Domain

besitzt er zwei unabhängige 8 b breite Schnittstellen, eine zum Schreiben neuer Zeichen und eine zweite für den VGA-Controller zum Lesen der anzuzeigenden Zeichen, da dieser auf keinen Fall warten darf, wenn der Pixeltakt eingehalten werden soll. Für eine einfache Adressierung werden die Spalten- und Zeilennummer zu einer 13 b breiten Adresse zusammengefügt. Das Verhalten des RAMs wird in einem generischen Modul beschrieben, das vom Compiler als RAM erkannt und mit Hilfe von BlockRAMs implementiert wird. Bei einer Datenbreite von 8 b und einer 13 b-Adresse ergibt sich ein Speicherbedarf von 64 kib, der vom Compiler mit zwei parallelen 36 kib-BlockRAMs erfüllt wird.

Für das Schreiben eines neues Zeichens existieren zwei Register, welche die Position speichern, an die das nächste Zeichen geschrieben wird. Dieser Cursor wird nach dem Schreiben eines Zeichens um einen Schritt vorgerückt, d. h. in die nächste Spalte oder an den Anfang der nächsten Zeile, falls er sich aktuell am Ende einer Zeile befindet. Dabei werden einige Steuerzeichen gesondert behandelt: Der Wagenrücklauf (CR, engl. carriage return) und Zeilenvorschub (LF, engl. line feed) sorgen dafür, dass der Cursor an den Anfang der aktuellen (CR) bzw. nächsten Zeile (LF) gesetzt wird. Die Rücktaste (BS, engl. backspace) setzt den Cursor auf das vorherige Zeichen und wird am Anfang einer Zeile ignoriert. Alle Steuerzeichen verändern stets nur den Cursor und werden nicht als Zeichen im Zeichenpuffer gespeichert.

Da in jedem Taktzyklus ein neues Zeichen geschrieben und auf ein Wartesignal oder eine Warteschlange für die zu schreibenden Zeichen verzichtet werden soll und pro Takt immer nur ein Zeichen in den BlockRAM geschrieben wird, ist es nicht möglich, die aktuelle Zeile nach einem Zeilenwechsel komplett zu leeren. Dies würde dazu führen, dass alte Daten aus dem vorherigen Durchlauf des Ringpuffers weiterhin in der aktuellen und älteren nicht komplett überschriebenen Zeilen sichtbar sind. Eine denkbare Lösung ist die Verbreiterung der parallel zu verarbeitenden Daten, so dass z. B. eine komplette Zeile auf einmal geschrieben wird. Da Virtex-5-BlockRAMs jedoch maximal 36 b breite Daten speichern, wären statt zwei mindestens 18 parallele BlockRAMs nötig. Eine effizientere Lösung ist die Speicherung der Anzahl beschriebener Spalten für jede Zeile, wenn der Cursor diese Zeile verlässt. Zusammen mit der aktuellen Position des Cursors werden damit Zeichen, die (noch) nicht überschrieben wurden, bei der Anzeige ignoriert, auch wenn sie weiterhin im Speicher stehen. Die dafür nötige Speichermenge passt aufgrund der geringen Datenbreite (7 b für maximal 80 Zeichen) in einen einzigen zusätzlichen BlockRAM.

Weiterhin implementiert das Terminal einen Scroll-Mechanismus. Dieser sorgt dafür, dass die Zeile, auf der aktuell der Cursor sitzt, immer am unteren Bildschirmrand angezeigt wird und alte Eingaben langsam nach oben verschoben werden, bis sie am oberen Bildschirmrand „verschwinden“, d. h. von der aktuellen Zeile überschrieben werden. Dies ist über eine Transformation der Zeilennummer beim Auslesen der Zeichen für die Anzeige implementiert, die eigentliche Speicherung in Form des Ringpuffers wird nicht verändert. So wird zu der aus den Pixelkoordinaten abgeleiteten Zeilennummer die aktuelle Zeile des Cursors und 1 addiert und anschließend der Rest bezüglich der Gesamtanzahl Zeilen (60) berechnet. Das sorgt dafür, dass die Zeile mit dem Cursor immer die letzte Zeilennummer besitzt.

4.2.4 TileLink-Integration

Das Terminal ist als VHDL-Modul implementiert. Das Interface besteht zum einen aus den verschiedenen DVI-Signalen und den beiden Leitungen für das I²C-Interface des CH7301C-DVI-Encoders und zum anderen aus einem Schreib-Port mit valid-Signal für das Schreiben von Zeichen. Ein ready-Signal wird hierbei nicht benötigt, da das Modul, wie oben beschrieben, in der Lage ist, in jedem Takt ein neues Zeichen zu schreiben.

Um das Terminal als MMIO-Gerät in den Adressraum des Prozessors einzubinden muss ein TileLink-Slave für das Terminal implementiert werden. Dazu wurde der in Rocket-Chip bereits enthaltene `TLRegisterRouter` verwendet, der die Implementierung für Register-basierte Slaves deutlich erleichtert. So müssen damit nur die Startadresse und die relativen Adressen der einzelnen Register definiert werden. Für jedes Register wird außerdem ein Schreib- und/oder ein Leseport angegeben, wobei es sich dabei um normale ready/valid-Schnittstellen handelt. Der `TLRegisterRouter` verteilt anschließend eingehende Anfragen auf die korrekten Register und generiert die passenden TileLink-Antworten. Außerdem lassen sich damit einfach Interrupts definieren und verwenden, was jedoch für das Terminal nicht nötig ist.

Um das Terminal in den Chisel-Code einzubinden wird dort eine BlackBox mit den selben Modul- und Signalnamen wie das VHDL-Modul definiert. Die DVI-Signale werden an das TopLevel-Chisel-Modul weitergeleitet, wo sie als TopLevel-Signale mit Hilfe der Constraint-Datei mit den passenden Pins des FPGAs verbunden werden. Da der Prozessor und das Terminal in verschiedenen Taktdomänen laufen, dürfen die Schreib-Ports des Terminals und des MMIO-Registers nicht direkt verbunden werden. Daher

wird dazwischen eine asynchrone Warteschlange mit der Länge eins eingefügt, deren Schnittstellen zum Einfügen und Entnehmen eines Wertes mit unterschiedlichen Taktraten angesprochen werden.

Das MMIO-Register für das Terminal wird als 32 b breites Register definiert, wobei die untersten 8 b beim Schreiben direkt in die Warteschlange eingefügt werden. Falls sich dort bereits ein Wert befindet, wird der neue Wert ignoriert. Dieses Phänomen würde selbst dann noch auftreten, wenn der Takt der entnehmenden Seite deutlich höher als der der einfügenden Seite wäre, da die Synchronisation der unterschiedlichen Takte innerhalb der asynchronen Warteschlange einige Takte Latenz erzeugt. Daher wird beim Lesen des kompletten Registers das höchstwertigste Bit auf 1 gesetzt, falls sich aktuell ein Wert in der Warteschlange befindet, womit ein Treiber vor dem Schreiben den aktuellen Status prüfen kann.

4.3 SD-Karte

Als Bootmedium kommt eine Mikro-SD-Karte zum Einsatz, da der in Freedom enthaltene Bootloader bereits einen einfachen Treiber dafür beinhaltet. Dieser verwendet das SPI zur Kommunikation mit der Karte. Nach dem Anlegen der korrekten Spannung (3,3V) benötigt der Mikrocontroller auf der Karte bis zu 1 ms und 74 Takte für die Initialisierung und befindet sich anschließend im SD-Modus [2, S. 203]. Zum Umschalten auf den SPI-Modus wird ein Reset-Kommando (CMD0) an die Karte übertragen. Dabei wählt der SPI-Controller die Karte, wie bei SPI üblich, mittels des Chip Select-Signals aus, wodurch die Karte das SPI-Protokoll erkennt und bis zum nächsten Spannungsverlust in den SPI-Modus wechselt. Anschließend werden noch einige weitere Initialisierungs- und Diagnosekommandos übertragen, bevor mit dem Lesen des Bootimages begonnen wird.

Das Lesen der Daten erfolgt bei SDHC- und SDXC-Karten[3] immer in 512 B großen Blöcken. Es gibt Kommandos zur Übertragung eines einzelnen Blocks (CMD17) oder aller Blöcke ab einer Startadresse (CMD18 zum Starten, CMD12 zum Stoppen). Nach jedem Block wird außerdem eine 2 B lange CRC-Prüfsumme übertragen.

[3] Alle Karten mit einer Kapazität größer als 2 GiB

Da das ML507-Entwicklungsboard nur einen Steckplatz für CompactFlash-Karten bietet, wurde für diese Arbeit mit Hilfe von KiCad und vorhandenen Komponenten eine Breakout-Platine entwickelt und anschließend auf einer selbst geätzten Platine aufgebaut (siehe Abbildung 4.1). Diese Platine ermöglicht es, die Mikro-SD-Karte auf ein normales Steckbrett zu stecken und die passenden Pins per Jumper-Kabel mit dem Entwicklungsboard zu verbinden.

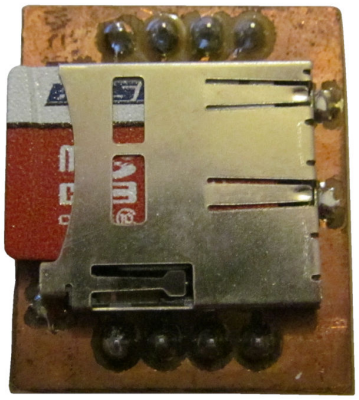

Abbildung 4.1: Breakout-Platine für Mikro-SD-Karten mit eingesteckter Karte

5 Software

5.1 Bootloader

Der in Freedom enthaltene Bootloader besteht aus einem Initialisierungsteil in RISC-V-Assembler und dem in C implementierten SD-Karten-Bootloader. Der Assembler-Teil pausiert als erstes in einem System mit mehreren Hardwarethreads alle Threads bis auf den ersten und initialisiert den Stackpointer (SP). Als nächstes wird der C-Code aufgerufen, welcher die UART- und SPI-Schnittstellen initialisiert. Über UART werden Debug-Nachrichten und eventuelle Fehler beim Booten ausgegeben und über SPI wird, wie in Abschnitt 4.3 beschrieben, die SD-Karte angesprochen und von dort das zu bootende Betriebssystem geladen. Dabei werden immer die ersten 16 MiB der SD-Karte an den Beginn des physischen Arbeitsspeichers kopiert. Anschließend wird der Kontrollfluss wieder an den Assembler-Code übergeben und alle vorher pausierten Hardwarethreads gestartet. Jeder Thread springt dann direkt an den Anfang des physischen Speichers, wobei als Argumente die eigene Thread-Id und ein Zeiger auf den DeviceTree in Registern übergeben werden.

5.2 DeviceTree

DeviceTree [9] ist eine Spezifikation zur Beschreibung der in einem eingebetteten System vorhandenen Hardware und wird u. a. von Linux dazu verwendet, das Betriebssystem zu initialisieren und die passenden Treiber zu laden. Dies ermöglicht es, ähnlich wie die bei PC-Systemen übliche Autokonfiguration, das gleiche kompilierte Linux-Image auf unterschiedlichen SoCs zu verwenden, solange die DeviceTrees die jeweilige Hardware korrekt beschreiben. Damit dies gewährleistet ist, sollte der DeviceTree ein Teil der

Hardware selbst sein. Bei der Freedom-Plattform wird dies durch die direkte Einbindung des binär codierten DeviceTree in den Bootloader-Code erreicht.

Der DeviceTree in Listing 5.1 auf Seite 34 ist eine stark verkürzte und damit nicht mehr funktionsfähige Version des DeviceTree für das in dieser Arbeit entwickelte SoC. Er wird automatisch beim Kompilieren des Chips von Rocket Chip erzeugt, musste jedoch für die Verwendung mit dem neuesten Freedom SDK um die fehlende Definition der Taktraten für die UART- und SPI-Schnittstellen erweitert werden. Außerdem musste ein globaler Alias für den seriellen Port hinzugefügt werden. Dieser stellt bei mehreren vorhandenen Ports eine konsistente Nummerierung sicher. In beiden Fällen werden die Komponenten indirekt über einen frei definierbaren Namen referenziert. Beim Kompilieren des im Textformat vorliegenden DeviceTrees in das Binärformat mit Hilfe des `dtc`-Programms werden diese Referenzen automatisch aufgelöst.

Die Auflistung der CPUs enthält nur einen Eintrag, da es sich um ein Single-Core-System handelt. Das `reg`-Attribut enthält im Falle der CPU die Hardware-ID des Threads, bei allen anderen Komponenten die Startadresse und Länge der zugewiesenen Speicherbereiche. Das `compatible`-Attribut dient der Zuordnung der passenden Treiber, welche darüber auch Zugriff auf die weiteren Attribute haben. So verwendet der Treiber für die serielle Schnittstelle die Informationen über die Interrupts, um beim Treiber des Interruptcontrollers die korrekten Interrupts zu registrieren. Der referenzierte Takt wird dagegen für die Berechnung des nötigen Takt-Teilers für die gewünschte Baudrate verwendet.

5.3 Linux

Als Betriebssystem für die Workstation auf dem ML507-Board kommt GNU/Linux zum Einsatz. Dazu wird das Freedom Unleashed-SDK [24] verwendet. Dieses beinhaltet das Buildroot-Programm, das Linux und in eingebetteten Systemen oft verwendete Programme automatisch für eine andere Architektur cross-kompiliert, d. h. die Kompilierung muss nicht auf einem System mit der selben Architektur stattfinden. Anschließend verpackt Buildroot den kompilierten Kernel und die ausgewählten Programme in ein bootfähiges Image. Dieses wird schließlich zusammen mit dem Berkeley Boot Loader (BBL) zum endgültigen Bootimage verpackt, das direkt auf die SD-Karte geschrie-

ben wird. Der BBL implementiert Gleitkommaoperationen in Software für Prozessoren ohne Hardwareimplementierung und zeigt optional ein textbasiertes Bootlogo an.

Linux unterstützt seit Version 4.15 offiziell RISC-V [8]. Dies beinhaltet den grundlegenden Initialisierungscode, die DeviceTree-Integration und Treiber für die virtuelle Speicherverwaltung und die Interrupt-Verarbeitung. Die GNU Compiler Collection und die GNU C-Standardbibliothek (glibc) unterstützen ebenfalls bereits seit einiger Zeit offiziell RISC-V [11]. Allerdings sind die Treiber für die Rocket Chip-Hardwarekomponenten noch nicht in Linux enthalten, so dass eine auf 4.15 basierende, für die Unleashed-Plattform angepasste Linux-Version verwendet wird. Diese enthält u. a. Treiber für SPI, UART und die GPIO-Pins.

5.3.1 UART-Treiber

Der UART-Treiber enthält neben der primären UART-Implementierung auch zwei Implementierungen der Linux-Konsolenschnittstelle. Damit lässt sich die Workstation über die serielle Schnittstelle von einem externen Computer aus bedienen. Die erste Implementierung dient ausschließlich der Anzeige von Log-Nachrichten während des frühen Bootvorgangs und funktioniert ohne den UART-Treiber. Nachdem dieser später im Bootvorgang erfolgreich geladen wurde, wird auf die reguläre UART-Konsole umgeschaltet, die neben der Ausgabe auch die Eingabe von Zeichen unterstützt.

Diese zweite Konsole funktionierte jedoch zu Beginn nicht korrekt. Zwar wurde sie, den Logs nach zu urteilen, ordnungsgemäß geladen und initialisiert, allerdings gab es nach Abschalten der frühen Bootkonsole (earlycon) keine weiteren Ausgaben. Da die earlycon jedoch lange genug aktiv war, konnte der Fehler vom Autor über Debug-Ausgaben an kritischen Stellen der Treiberinitialisierung identifiziert werden. Es stellte sich heraus, dass der UART-Treiber registriert wurde, bevor die Konsole komplett initialisiert war. Linux versuchte allerdings direkt nach der Registrierung die zugehörige Konsole zu aktivieren, was damit scheiterte. Es ist vorstellbar, dass dieses Problem bei einem Mehrkernsystem aufgrund einer anderen Ausführungsreihenfolge der Kernel-Threads nicht auftritt, jedoch konnte dies nicht verifiziert werden. Ein Patch mit einer Korrektur für die Initialisierungsreihenfolge wurde an den RISC-V-Linux-Port übermittelt [21].

5.3.2 Terminal-Treiber

Für das Terminal wurde ein neuer Treiber entwickelt. Als Vorlage diente dabei der Konsolen-Treiber für den Android-Emulator (Goldfish), da dieser DeviceTree-kompatibel ist und ausschließlich ein auf MMIO-Registern basierendes Terminal implementiert. Die Behandlung von Interrupts und Leseanfragen wurde entfernt, da das Workstation-Terminal keine Eingaben zurückliefert. Außerdem wurde der DMA-basierte Schreibvorgang, bei dem nur ein Zeiger auf die zu schreibenden Daten in ein Register geschrieben wird, durch ein direktes, sequenzielles Schreiben der einzelnen Zeichen ersetzt. Einige Details zum korrekten Auslesen der MMIO-Adresse aus dem DeviceTree wurden aus dem im vorigen Abschnitt beschriebenen UART-Treiber übernommen.

Der Terminal-Treiber implementiert ebenfalls eine earlycon und eine reguläre Konsole. Erstere funktioniert aktuell nicht, obwohl die Implementierung bis auf die Namen der Funktionen identisch zu denjenigen im UART- und Goldfish-Treiber ist. Die reguläre Konsole funktioniert dagegen problemlos, allerdings kann aktuell immer nur eine Konsole gleichzeitig aktiviert werden. Mehrere Konsolen werden normalerweise über die Linux-Kommandozeile folgendermaßen angegeben.

```
CONFIG_CMDLINE="console=ttyML0 console=ttySI0,115200"
```

Die Unterstützung von Kommandozeilenparametern im RISC-V-Port von Linux ist jedoch aktuell noch unvollständig, so dass diese Angaben ignoriert werden. Daher kann das Terminal zwar als Konsole genutzt werden, allerdings können in diesem Fall keine Zeichen eingegeben werden, da der UART-Treiber nicht geladen ist. Alternativ kann das Terminal von normalen Programmen als Ausgabe verwendet werden, während die Steuerung über UART erfolgt.

```
/ {
    model = "freechips,rocketchip-unknown";
    aliases {
        serial0 = &L8;
    };
    L15: cpus {
        L5: cpu@0 {
            device_type = "cpu";
            compatible = "sifive,rocket0", "riscv";
            mmu-type = "riscv,sv39";
            riscv,isa = "rv64imafdc";
            reg = <0>;
            clock-frequency = <60000000>;
            next-level-cache = <&L12>;
        };
    };
    L12: memory@80000000 {
        device_type = "memory";
        reg = <0x80000000 0x10000000>;
    };
    L14: soc {
        sysclk: sysclk {
            compatible = "fixed-clock";
            clock-frequency = <60000000>;
        };
        L0: interrupt-controller@c000000 {
            interrupt-controller;
            compatible = "riscv,plic0";
            reg = <0xc000000 0x4000000>;
        };
        L6: rom@10000 {
            compatible = "sifive,maskrom0";
            reg = <0x10000 0x2000>;
        };
        L8: serial@64000000 {
            compatible = "sifive,uart0";
            reg = <0x64000000 0x1000>;
            clocks = <&sysclk 0>;
            interrupt-parent = <&L0>;
            interrupts = <1>;
        };
        L9: terminal@64003000 {
            compatible = "klemens,terminal0";
            reg = <0x64003000 0x1000>;
        };
    };
};
```

Listing 5.1: Verkürzter DeviceTree für Linux auf der ML507-Workstation

6 Zusammenfassung

Das Ziel dieser Arbeit war die Implementierung einer freien RISC-V-Workstation auf dem ML507-Entwicklungsboard mit Unterstützung für einen selbstständigen Linux-Bootprozess. Außerdem sollte die Workstation Eingaben von einer Tastatur entgegennehmen und Ausgaben per DVI auf einem Bildschirm anzeigen.

Der erste Teil der Aufgabe wurde auf Basis der Freedom Unleashed-Plattform von SiFive umgesetzt. So verwendet die fertige Workstation einen von Rocket Chip erstellten SoC mit dem in der modernen Hardwarebeschreibungssprache Chisel geschriebenen RISC-V-Prozessor Rocket. Dieser besitzt alle für den Linux-Betrieb notwendigen Eigenschaften wie die Unterstützung von virtuellem Speicher und eine Implementierung der RISC-V Privileged Architecture. Außerdem enthält das SoC einen BootROM mit einem Bootloader, der über SPI von einer SD-Karte ein Betriebssystem lädt und ausführt. Damit ist die Workstation in der Lage, komplett selbstständig zu booten. Für den Anschluss der SD-Karte wurde außerdem eine Breakout-Platine entworfen und gebaut, da das ML507-Board keine entsprechende Schnittstelle besitzt.

Rocket musste für den älteren Virtex-5-FPGA auf dem nicht mehr unterstützten ML507-Board deutlich verkleinert werden, damit er zusammen mit den übrigen Komponenten auf dem FPGA Platz findet. Deswegen wird nur ein einziger Rocket-Kern mit verkleinerten Caches eingesetzt. Die Anbindung an den auf dem ML507-Board vorhandenen Arbeitsspeicher erfolgte mit Hilfe des proprietären MIG-Cores, der von Xilinx für das Entwicklungsboard bereitgestellt wird. Dieser wurde zuerst in einem VHDL-Modul gekapselt, um das proprietäre Interface zu verbergen, und anschließend über einen in Chisel implementierten TL-Slave an das TileLink-Speichersystem von Rocket angeschlossen.

Für die Ausgabe wurde ein Terminal in VHDL implementiert, das bis zu 80x60 Zeichen auf einem per DVI angeschlossenen Monitor darstellt. Es unterstützt dabei grundlegende Steuerzeichen wie den Zeilenumbruch oder die Rücktaste. Außerdem wurde für das

Terminal ein Linux-Treiber entwickelt, wobei dieser aktuell aufgrund von Einschränkungen des RISC-V-Linux-Ports nicht sinnvoll als Konsole nutzbar ist. Die Eingabe von Zeichen über eine Tastatur konnte dagegen nicht umgesetzt werden. Zwar wurde bereits ein VHDL-Modul für das Auslesen der Tastencodes einer an das Entwicklungsboard angeschlossenen Tastatur entwickelt, für die Integration in das Rocket-SoC und die Entwicklung eines neuen oder die Anpassung eines vorhandenen Linux-Treibers blieb jedoch keine Zeit mehr. Stattdessen wurde die auf dem ML507-Board vorhandene serielle Schnittstelle an das SoC angebunden, was aufgrund vorhandener Hardwaremodule und Linux-Treiber schneller umgesetzt werden konnte. Damit lässt sich die Workstation über einen per UART angeschlossenen Computer bedienen.

Die fertige Workstation läuft, abgesehen von den erwähnten Einschränkungen, zuverlässig. Das auf 16 MiB beschränkte Bootimage begrenzt aktuell noch die mögliche Softwareauswahl, allerdings ließe sich der Bootloader so anpassen, dass auf der SD-Karte ein normales Partitionslayout verwendet werden kann. Damit wäre zukünftig auch die Verwendung einer etablierten Linux-Distribution wie Debian möglich. Eine weitere nützliche Erweiterung wäre die Anbindung eines generischen Peripherie-Bussystems wie PCIe oder USB. Damit ließen sich eine Vielzahl weiterer Geräte wie Netzwerkschnittstellen und Eingabegeräte direkt verwenden. Schließlich wäre auch eine weitere Optimierung des Rocket-SoCs durch Vergrößerung der Caches oder einer Steigerung des Taktes denkbar, denn das FGPA ist aktuell nur zu knapp 90 % ausgelastet.

Sämtliche für die Workstation vom Autor entwickelten Module und alle externen Komponenten bis auf den MIG-Core befinden sich, unter GPLv3 lizenziert, in einem gemeinsamen git-Repository [20]. Der MIG-Core muss dabei selbst erstellt werden, da die Lizenz von Xilinx keine Weiterverteilung erlaubt. Das Repository enthält außerdem eine Anleitung zum Kompilieren der Workstation für das Ml507-Entwicklungsboard.

Literatur

[1] Krste Asanović u. a. *The Rocket Chip Generator*. Techn. Ber. UCB/EECS-2016-17. EECS Department, University of California, Berkeley, Apr. 2016. URL: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[2] SD Card Association. *SD Specifications. Physical Layer Simplified Specification*. Version 6.00. 10. Apr. 2017. URL: https://www.sdcard.org/downloads/pls/index.html.

[3] J. Bachrach u. a. "Chisel: Constructing hardware in a Scala embedded language". In: *DAC Design Automation Conference 2012*. Juni 2012, S. 1212–1221. DOI: 10.1145/2228360.2228584.

[4] Papon Charles. *VexRiscv*. URL: https://github.com/SpinalHDL/VexRiscv (besucht am 05. 02. 2018).

[5] Chrontel. *CH7301C DVI Transmitter Device*. URL: http://www-inst.eecs.berkeley.edu/~cs150/sp13/resources/CH7301C.pdf.

[6] OpenRISC Community. *OpenRISC*. URL: https://openrisc.io/ (besucht am 01. 06. 2018).

[7] Henry Cook, Wesley Terpstra und Yunsup Lee. "Diplomatic Design Patterns: A TileLink Case Study". In: *Workshop on Computer Architecture Research with RISC-V*. 2017. URL: https://carrv.github.io/2017/papers/cook-diplomacy-carrv2017.pdf.

[8] Palmer Dabbelt. *All Aboard, Part 8: The RISC-V Linux Port is Upstream!* 5. Dez. 2017. URL: https://www.sifive.com/blog/2017/12/05/all-aboard-part-8-the-risc-v-linux-port-is-upstream/ (besucht am 04. 06. 2018).

[9] *Devicetree Specification*. Version 0.2. 20. Dez. 2017. URL: https://github.com/devicetree-org/devicetree-specification/releases/tag/v0.2.

[10] Jordan Earls. *Using ISE's new parser for VHDL on old hardware*. 8. Sep. 2012. URL: http://earlz.net/view/2012/09/08/1526/using-ises-new-parser-for-vhdl-on-old (besucht am 16. 05. 2018).

[11] RISC-V Foundation. *Software Status*. URL: https://riscv.org/software-status/ (besucht am 04. 06. 2018).

[12] RISC-V Foundation. *The RISC-V Instruction Set Manual. Volume I: User-Level ISA*. Hrsg. von Andrew Waterman und Krste Asanović. Version 2.2. 7. Mai 2017. URL: https://riscv.org/specifications/.

[13] RISC-V Foundation. *The RISC-V Instruction Set Manual. Volume II: Privileged Architecture*. Hrsg. von Andrew Waterman und Krste Asanović. Version 1.10. 7. Mai 2017. URL: https://riscv.org/specifications/privileged-isa/.

[14] Digital Display Working Group. *Digital Visual Interface. DVI*. Version 1.0. 2. Apr. 1999. URL: https://web.archive.org/web/20120717013308/http://www.ddwg.org/lib/dvi_10.pdf.

[15] Nathan Ickes. *VGA Video*. URL: http://web.mit.edu/6.111/www/s2004/NEWKIT/vga.shtml (besucht am 14. 01. 2018).

[16] National Instruments. *Anatomy of a Video Signal*. 6. Dez. 2006. URL: http://www.ni.com/white-paper/3020/en/ (besucht am 02. 06. 2018).

[17] JEDEC. *DDR2 SDRAM SPECIFICATION*. JESD79. Version 2E. Apr. 2008.

[18] Sergey Khabarov. *RISC-V River Core*. URL: https://github.com/sergeykhbr/riscv_vhdl (besucht am 18. 05. 2017).

[19] UC Berkeley Architecture Research. *Berkeley Out-of-Order Machine*. URL: https://github.com/ucb-bar/riscv-boom (besucht am 03. 06. 2018).

[20] Klemens Schölhorn. *RISC-V Linux-Workstation auf dem ML507*. Commit-Hash: 05df1a9d8f91f4e54f455c34dd8dfe4098643591. URL: https://git.tiband.de/riscv/workstation.

[21] Klemens Schölhorn. *sifive-serial: register uart port as console before adding it*. 20. Mai 2018. URL: https://github.com/riscv/riscv-linux/pull/137 (besucht am 04. 06. 2018).

[22] SiFive. *Freedom*. URL: https://github.com/sifive/freedom (besucht am 03. 06. 2018).

[23] SiFive. *Freedom Platforms*. URL: https://www.sifive.com/products/freedom/ (besucht am 03. 06. 2018).

[24] SiFive. *Freedom Unleashed Software Development Kit*. URL: https://github.com/sifive/freedom-u-sdk (besucht am 04. 06. 2018).

[25] SiFive. *TileLink Specification*. Version 1.7-draft. 22. Aug. 2017. URL: https://static.dev.sifive.com/docs/tilelink/tilelink-spec-1.7-draft.pdf.

[26] Xilinx. *AR 29313. When changing a MIG-generated Virtex-5 FPGA DDR2 SDRAM pin-out, both the UCF and top-level HDL parameters MUST be updated*. 7. Apr. 2009. URL: https://www.xilinx.com/support/answers/29313.html (besucht am 08. 04. 2013).

[27] Xilinx. *LogiCORE IP AXI Bridge for PCI Express*. URL: https://www.xilinx.com/support/documentation/ip_documentation/axi_pcie/v1_04_a/pg055-axi-bridge-pcie.pdf.

[28] Xilinx. *Memory Interface Solutions User Guide*. UG086. Version 3.6. 21. Sep. 2010. URL: https://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf.

[29] Xilinx. *ML505/ML506/ML507 Evaluation Platform User Guide*. UG347. Version 3.1.2. 16. Mai 2011. URL: https://www.xilinx.com/support/documentation/boards_and_kits/ug347.pdf.

[30] Xilinx. *ML507 Master UCF Pin Constraints*. URL: http://www.xilinx.com/products/boards/ml505/ml505_12.1/docs/ml50x_U1_fpga.ucf (besucht am 22. 08. 2015).

[31] Xilinx. *Product Discontinuation Notice for Development Systems Products*. XCN14010. Version 2.0. 23. März 2015. URL: https://www.xilinx.com/support/documentation/customer_notices/xcn14010.pdf.

Selbstständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit selbstständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Leipzig, 6. Juni 2018,