

Softprozessoren

FPGA Development Teil II

Sven Gregori, CN8
<gregori@hs-furtwangen.de>

UnFUG WS 08/09
Hochschule Furtwangen

6. November 2008



Einführung / Rückblick



Embedded System

- Special Purpose Computersystem (mehr oder weniger)
- System ist Teil des Systems :p
- Ressourcen (wenn möglich) auf die Anwendung angepasst
- Beispiele
 - Mobiltelefon, DVD Player, Getränkeautomat, HW Router, Drucker, Mikrowelle, Navi, Fernbedienung, ...



Mikrocontroller

- Single Chip Computer
- CPU und zusätzliche Komponenten auf einem Chip
 - USB Host Controller, Ethernet MAC, Memory Controller, GPIO, ADC, DAC, RAM, ROM, ...
- kleinste (Embedded) Systeme benötigen ausser etwas analoger Elektronik keine weiteren Komponenten (siehe Mikrocontrollerbasteleien letztes Semester)



FPGA

- *Field Programmable Gate Array*
- frei programmierbarer / rekonfigurierbarer Logikbaustein
- "Programmierung" definiert eigentliches Verhalten
- ermöglicht eigene Digitalbausteine zu entwerfen



Vorteile

- System der Anwendung anpassen
- echte Parallelität
- Rekonfigurierbarkeit
 - Designänderungen ohne Hardware anzufassen
 - schnelles ASIC Prototyping
- Kostenfaktor gegenüber ASIC bei geringen Stückzahlen



Nachteile

- je nach Situation ist Software Lösung sinnvoller
- benötigt meistens externen Speicher für Konfiguration
- je nach FPGA nur relativ geringe Taktraten möglich
- Kostenfaktor gegenüber ASIC bei höheren Stückzahlen



Umsetzung

- "Programmierung" auf abstrakter Ebene
 - Hardware Description Language
 - VHDL, Verilog, SystemC, ...
- Design testen
 - Testbench
 - Simulation
- Constraints definieren
 - Pinbelegung, Timing, Placement, ...



Umsetzung

- Synthese
 - Übersetzen von HDL Code
 - Abbilden in FPGA-spezifische Elemente
 - Darstellung in Form von sog. Netzliste
- Implementierung
 - Netzliste den tatsächlichen Komponenten innerhalb des FPGA zuordnen und diese verbinden
 - Bitstream Erzeugung
- Bitstream konfiguriert FPGA



IP Core

- fertige Komponente
- Vielfalt bereits existierender IP Cores
 - können ins eigene Design mit aufgenommen werden
 - als Source Code oder Netzliste
 - kommerzielle und Open Source
 - opencores.org
- "beliebig viele" Cores können in ein FPGA gepackt werden
 - abhängig von verfügbaren Ressourcen



Beispiele

- Arithmetikeinheit
- UART
- Ethernet MAC
- USB Host Controller
- Crypto Core
- ...
- Softprozessor <o/

Softprozessoren



Softprozessoren

- Prozessor in (z.B) einem FPGA implementiert
- mit allen Vorteilen aus der konfigurierbaren Welt
- sprich Prozessor kann den Wünschen und dem Zielsystem angepasst werden
- zusammen mit anderen Komponenten erhält man so einen anwendungsspezifischen Mikrocontroller



Konfigurierbarkeit

- theoretisch **alles** einstellbar
- interne Komponenten
 - Cache, Arithmetik, FPU, MMU, ...
- Busbreiten
- Endianess
- Registersatz
- Instruction Set



Instruction Set

- Prozessor mit Anwendungsspezifischen Opcodes ausstatten
 - Hardware Implementierungen in Software mappen
- damit nicht auf CPU beschränkt
 - GPU, DSP, Math/Crypto/Biotech/... Prozessor denkbar
- Problem: benötigt entsprechenden Compiler



Vorteile

- CPU nach eigenem Wunsch zusammenstellen
- vom internem Aufbau bis hin zum Instruction Set
- Anwendungsspezifisches System als Mikrocontroller auf einem Chip realisierbar (SoPC)
- möglicher Kompromiss bei der Frage Software oder Hardware
- Spassfaktor ;)



Nachteile

- echte CPU im direkten Vergleich in der Regel performanter
- relativ geringe Taktfrequenz (je nach FPGA)
- Kosten



cw667x

- Brainfuck CPU \o/
- führt nativ Brainfuck Code aus
- Instruction Set folglich die bekannten 8 Brainfuck Befehle
- entwickelt von Clifford Wolf als Beispielprojekt zum Thema VHDL Development für den 20C3



JOP

- *Java Optimized Processor*
- führt nativ Java Bytecode aus
- Open Source Core



OpenSPARC

- Sun Microsystems
- Open Source Core
- UltraSPARC T1 und T2



OpenCores.org

- OpenRISC
- 8051 Clones
- AVR Clones
- 68k Clones
- zich andere



FPGA Herstellerspezifische

- Altera Nios II
- LatticeMico32
 - Open Source
- Xilinx PowerPC 440/405
 - keine Softprozessoren sondern "harte" Kerne
 - als Resource in speziellen Xilinx Virtex FPGAs vorhanden
- Xilinx MicroBlaze
 - gleich mehr

Xilinx MicroBlaze



MicroBlaze

- 32Bit RISC Architektur
- als Netzliste in *Xilinx EDK* verfügbar
- VHDL Source Code Lizenz- und Kostenpflichtig :/



Konfigurierbarkeit

- Data und Instruction Cache Größen
- MMU (seit MicroBlaze 7.0 - aktuell 7.10.d)
 - ...
- Single Precision Floating Point Unit
- Pipeline
 - 3stufig - Flächenoptimierung
 - 5stufig - Geschwindigkeitsoptimierung
- Hardware Multiplizierer (disabled/32Bit/64Bit)
- Hardware Dividierer
- Barrel Shifter
- Exceptions



Xilinx EDK

- *Xilinx Embedded Development Kit*
- beinhaltet *Xilinx Platform Studio*
- Unterstützung aller (?) Xilinx Development Boards
- neben CPU noch andere Komponenten definierbar und konfigurierbar (Speicher, UART, Timer, Ethernet MAC, ...)
- Konfiguration von Busanschlüssen und Adressmapping
- System in ner GUI zusammenklicken
- im Hintergrund aber alles über Textfiles



Board Support Package

- Software Unterstützung für alle Komponenten
- Low Level Treiber
- besteht aus Konfigurationsdatei und TCL Script
 - in der Regel proprietärer Teil
- Details folgen gleich..



libxil

- Haupt-API
- beinhaltet die Low Level Treiber
- wird für Standalone Applikationen oder in Verbindung mit Betriebssystemen verwendet



Toolchain

- Portierung von *gcc*, *binutils* und *gdb* vorhanden
- jedoch nicht offiziell sondern Third Party Vendors
 - Xilinx (EDK)
 - PetaLogix
- Prozessorspezifische Einstellungen wie gewohnt mit `gcc -m<option>`
 - `gcc -mno-xl-soft-mul`
 - `gcc -mhard-float`
 - `gcc -mxml-soft-div`
 - ...



Standalone

- Anwendung direkt oberhalb der Hardware
- also kein Betriebssystem
- *libxil* steht zur Verfügung
- damit auch Low Level Treiber der Komponenten
- RS232 Komponente kann im BSP als stdin/stdout definiert werden, libxil bringt I/O Funktionen
- allerdings halt keine Grundfunktionen eines OS
 - Threads, Semaphoren, Message Queues, ...



Betriebssysteme für MicroBlaze

- Linux
 - uClinux (nur Kernel 2.4)
 - PetaLogix PetaLinux
 - (LynuxWorks BlueCat Linux)
- Xilkernel
- FreeRTOS
- eCos (bedingt - nicht offiziell)
- diverse kommerzielle RTOSs
 - ThreadX, uC/OS-II, Nucleus, ...



Xilkernel

- kommt auch mit *Xilinx EDK* mit
- Konfiguration über EDK (GUI) oder `system.mss`
- BSP generiert *libxilkernel*
→ `gcc ... -lxilkernel`
- Threads, Semaphoren, Shared Mem, Interrupt Handling, ...
- POSIX API
- recht kleiner Footprint



FreeRTOS

- minimales RTOS
- Standalone BSP, Konfiguration über `FreeRTOSConfig.h`
- paar Source Files, einfach mit dazukompilieren
- Memory Management wird über hinzunahme von `heap_1.c`, `heap_2.c` oder `heap_3.c` gewählt



eCos

- ziemlich umfangreiches RTOS
- verschiedene Pakete zur Auswahl
 - TCP Stack, Dateisysteme, USB Stack, ...
- allerdings keine offizielle MicroBlaze Unterstützung
- mONeCos als Projekt von Michal Šimek¹
 - leider noch sehr unausgereift und nicht wirklich für Produktiveinsatz brauchbar bisher
- eCos bringt *configtool* mit, BSP generiert Template dafür

¹MicroBlaze Linux Maintainer, PetaLinux Entwickler



PetaLinux

- Linux Distribution mit MicroBlaze Portierung (Kernel 2.6)
- uClinux mit dabei, inklusive Userland Applikationen
- BSP generiert `Kconfig.auto`
→ nach `arch/microblaze/platform/$platform` kopieren
- beinhaltet Treiber für Board Komponenten
 - bestehen aus `libxil` Low Level Treibern und `adapter.c` zur Einbindung der Treiber in Kernel API
- MMU Unterstützung in Entwicklung (Testrelease oder SVN)
- Distribution bringt Toolchain mit
- Kernelteil (natürlich) GPL, BSP eben proprietäre Lizenz



Demosystem



Board

- Xilinx XtremeDSP Starter Kit
- Xilinx Spartan-3A DSP 1800 FPGA
- 128MB DDR2 SDRAM
- 8MB SPI Flash
- 16MB Parallel Flash
- 10/100/1000 Ethernet PHY
- RS232
- VGA
- ADC, DAC
- ...



FPGA

- MicroBlaze CPU
- 32kB Data und Instruction Cache
- UART
- 8 LEDs
- DDR2 SDRAM Controller
- SPI Flash Controller
- Ethernet MAC
- Hardware Timer
- Interrupt Controller

SPI Flash

```
# cat /proc/mtd
dev:      size  erasesize  name
mtd0: 00100000 00010000 "bitstream"
mtd1: 00300000 00010000 "kernel"
mtd2: 00200000 00010000 "romfs"
mtd3: 00200000 00010000 "rwfs"
```

- FPGA lädt Konfiguration aus mtd0
- Bootloader (im FPGA BRAM) lädt Kernel Image aus mtd1
- Kernel lädt read-only Root Dateisystem aus mtd2
- read-write Partition mit JFFS2 Dateisystem auf mtd3



Sonstiges

- Bootloader um Kernel vom Flash in RAM zu laden
- PetaLinux, Kernel 2.6.20-uc0
- kleiner Webserver zum schlaue Sachen machen ;)
- MTD Treiber zur Flash Ansteuerung
 - verwendeter Flash Controller ermöglicht SPI Flash Adressraum in normalen CPU Adressraum zu mappen



Fazit



Fazit

- FPGAs allein sind ja schon ein nettes Spielzeug
- Softprozessoren eröffnen dem Ganzen aber nochmal ganz neue Dimensionen der Möglichkeiten und des Spasses



weitere Informationen

mein Kram

- *Embedded Systems*, UnFUG SS 2006
- *FPGA Development*, UnFUG SS 2008
- Diplomarbeit *Embedded Plattform auf FPGA Basis* (bald :p)

Links

Open Source IP Cores

- <http://www.opencores.org>

Xilinx MicroBlaze

- <http://www.xilinx.com/microblaze>

PetaLogix Developer Portal

- <http://developer.petalogix.com/>

Michal Šimeks Development Wiki

- <http://monstr.eu>



Fragen?