TECHNISCHE UNIVERSITÄT DRESDEN

Fakultät Informatik, Institut für Technische Informatik, Professur VLSI-Entwurfssysteme, Diagnostik und Architektur

Entwicklung einer parametrierbaren Steuer-Prozessor-Einheit für den Einsatz in Mixed-Signal ASICs

Vortrag zum Diplom

Stephan Richter - s3300266@mail.zih.tu-dresden.de

Dresden, 02.08.2011

Externes Unternehmen

DMOS Dresden MOS Design

DMOS GmbH
Bergstraße 4
01069 Dresden

Arbeitsgebiet

- Spezialisiert auf Herstellung von Mixed-Signal-Hochvolt-ASICs und ASSPs im Automobilbereich

ASSP : Anwendungsspezifisches Standardprodukt (mehrere Kunden)

Gliederung

01 Einführung (1)

Problemstellung

- Entwurf von dedizierter Hardware kostet viel Zeit und Geld
- Die Wiederverwendbarkeit bei applikativen Änderungen ist durch Metal-Redesigns gegeben aber aufwendig umzusetzen

- Lösungsansatz: Verlagerung von komplexen Steueralgorithmen in Software

- Entwurf einer Steuer-Prozessor-Einheit mit Multithreading
- Entwurf und Implementierung eines geeigneten Schedulingalgorithmus in Hardware

01 Einführung (2)

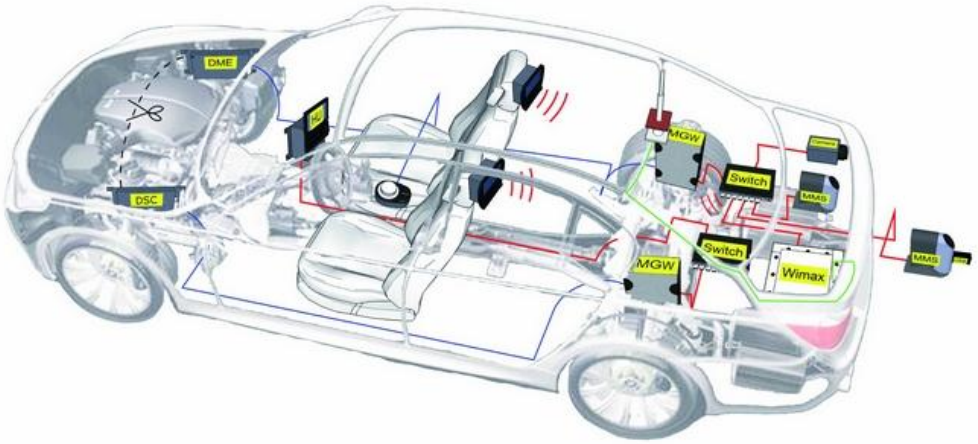
Anforderungen

- Multithreadingarchitektur unterstützt durch einen Hardwarescheduler
- Einheitlicher Adressraum für interne Register und externen Speicher
- Frei Parametrierbar
- „1-Takt-Pro-Befehl"-Machine → (Antipipelining)
- Energieeffiziente Architektur
- 8 MHz Taktfrequenz auf 0,36µ Standardzellen

01 Einführung (3)

Elektronik im Fahrzeug:

- Motorsteuerung
- Body Control Unit
- Einparkhilfe
- Fensterheber
- Komfortelektronik
- Lichtautomatik
- Motorkontrollleuchten
- Regensensor, Scheibenwischautomatik
- …

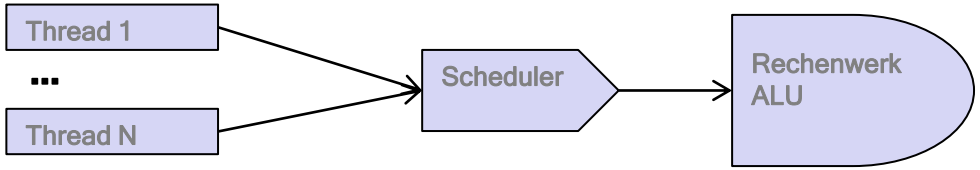


Quelle: [7]

➢ Die meisten Steuer /Überwachungsgeräte befinden sich den größten Teil der Fahrt im Sleep-Modus!

02 Hardwareseitiges Multithreading (1)

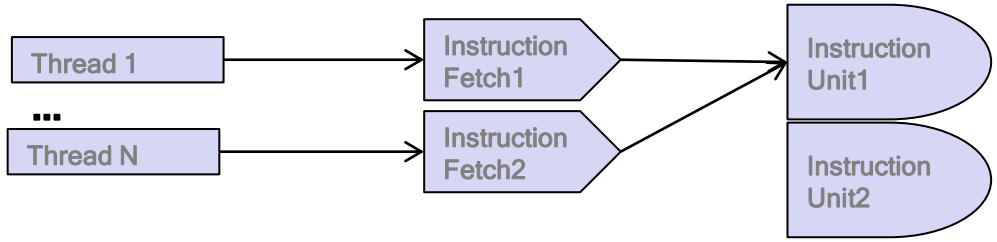
Fine-grained multithreading



- Threadwechsel per Round-Robin oder bei Pipeline-Hazards → Leerlauftakte
  - cache miss
  - branch misprediction
  - data dependency
- Nutzt nicht alle Ressourcen aus, da alle Ressourcen genau einem Thread zugeordnet sind

02 Hardwareseitiges Multithreading (2)

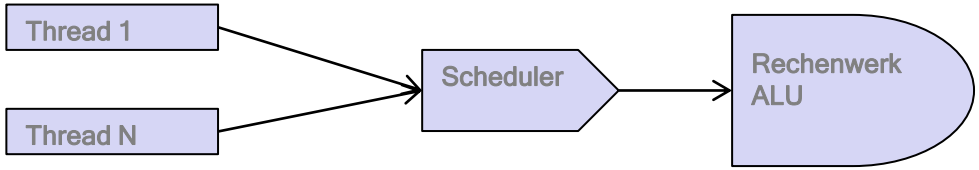
Simultaneous Multithreading / Hyper-Threading (Intel)



- Threads können gleichzeitig alle Ressourcen auszunutzen
- Beispiel: Intel Pentium 4 HT | Intel Core i7 Serie (30% SpeedUp behauptet Intel)
- Benötigt mehrere Instruction Fetch Units → mehrere Register zum Zwischenspeichern

02 Hardwareseitiges Multithreading (3)

Multithreading ohne Pipelining



- Threadwechsel per Scheduler und durch Prioritätssteuerung
- → Inaktive Threads verursachen keine Leerlauftakte
- Threadwechsel jederzeit möglich, da kein Piplining verwendet wird
- → Quasi-Parallele Abarbeitung

02 Hardwareseitiges Multithreading (4)

Einsatz von Multithreading in aktuellen eingebetteten Prozessoren

- Hat vor allem im Bereich der DSPs an Bedeutung gewonnen

Ubicom 3K und 5K Familie (8 - 10 Threads)

MIPS 34K Familie (5 Threads)
- 9 Stufige Pipeline

→ Keine Multithreading-Prozessoren ohne Pipelinestufen

03 Entwurf (1) - Parameter

Parameter

- Anzahl der Threads
- Registeranzahl pro Thread separat
- Stacktiefe der einzelnen Threads
- Adressbreite des externen Speicherbus (RAM)
- Breite der Register
- Breite des externen Speicherbus
- Schedulerparameter: Startpriorität, Interruptpriorität

03 Entwurf (2) - Befehlssatz

NOP	NOP	
CALL	12 Bit JMP	PC+1 -> Stack
JMPR	12 Bit rel Sprung	
BCZ (Branch)	PC +-= rA	zero
BCNZ	PC +-= rA	not zero
BCN	PC +-= rA	negative
BCNN	PC +-= rA	not negative
BCC	PC +-= rA	carry set
BCNC	PC +-= rA	carry not set
CALLI	PC = Reg[b]	PC+1 -> Stack
JMPI	PC = Reg[b]	
LD_PC	Reg[b] = PC+1	
RET	PC = Stack	
SET_PRIO	Setze eigene Priorität	
CLI	Scheduler aus	
SEI	Scheduler an	

ADD/SLL/ADC/ROL	Reg[b] += Reg[a] (+c)
SUB/SBC	Reg[b] -= Reg[a] (-c)
AND	Reg[b] &= Reg[a]
OR	Reg[b] |= Reg[a]
XOR	Reg[b] ^= Reg[a]
CP (compare)	Reg[b] - Reg[a]
CPI	Reg[b] - d
LD	Reg[b] = Reg[a]
ROR/SLR	Reg[b] << 0 (c)
INC	Reg[b]++
Dec	Reg[b]--
NOT	Reg[b] = !Reg[b]
LD_ID	Reg[0] = d
SETBIT	Reg[b][s] = 0
CLRBIT	Reg[b][s] = 1
CHKBIT	C-Flag = Reg[b][s]

- Alle Befehle in einem Takt abgearbeitet!
- Indirekte Adressierung bei arithmetischen Operationen möglich

03 Entwurf (3) - Scheduling

Mögliche Schedulingalgorithmen

- First In First Out
- Earliest Deadline First
- Least Slack Time First

➢ Diese Verfahren benötigen genaue Zeitinformationen und sind sehr aufwändig in Hardware umzusetzen

03 Entwurf (4) - Scheduling

Modifizierter Round Robin Scheduler

Beispiel mit **8 Threads**

Benötigtes Verhalten:

request(8)	select(8)	select_nxt(8)
1111 1111	oooo oo1o	oooo o1oo
1ooo o111	oooo o1oo	1ooo oooo
1111 oo1o	1ooo oooo	oooo oo1o

Gleichung für Linksschieben:

$$requestLEFT \Leftarrow \overline{(select - 1) \cup select} \cap request$$

$$selectLEFT \Leftarrow \overline{(requestLEFT + 1)} \cap requestLEFT$$

03 Entwurf (5) - Scheduling

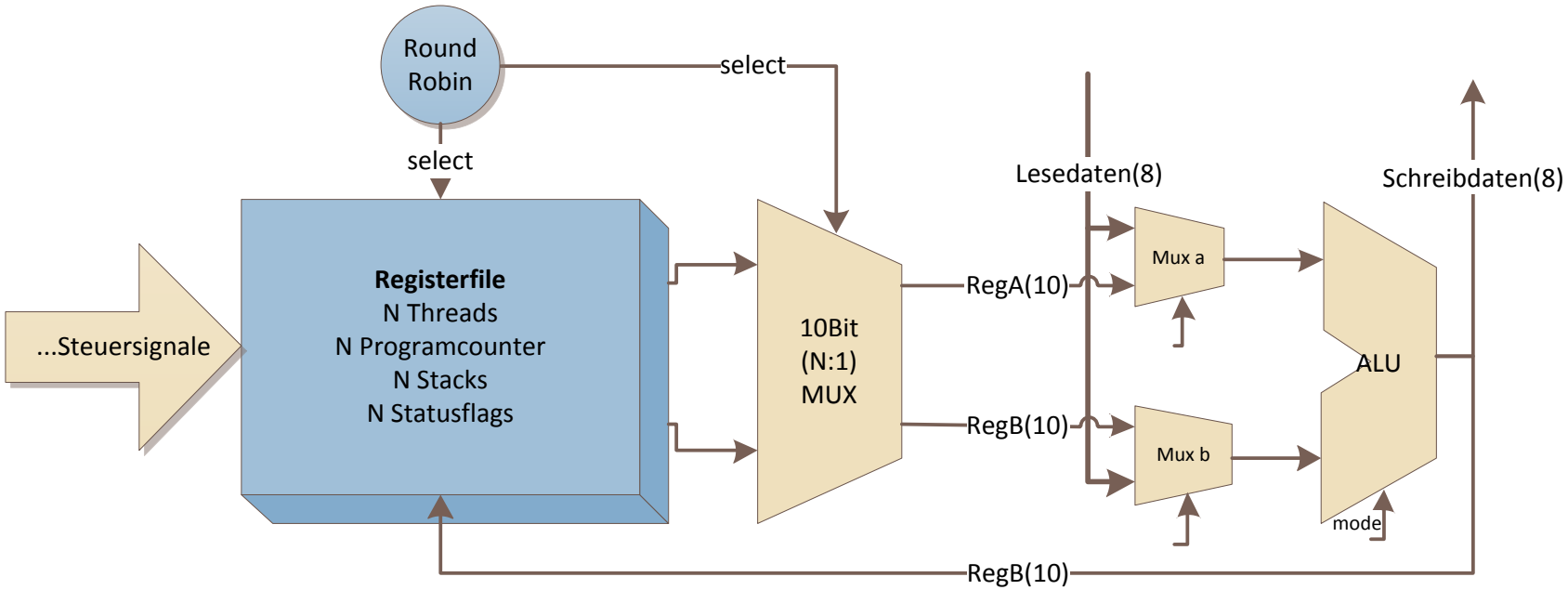
Die *request*-Signale ergeben sich aus rückwärtszählenden Counter beginnend bei der Priorität. Dieser setzt sich am Ende eines Zyklus wieder zurück.

Beispiel

4 Threads mit Prioritäten (5,2,2,1)

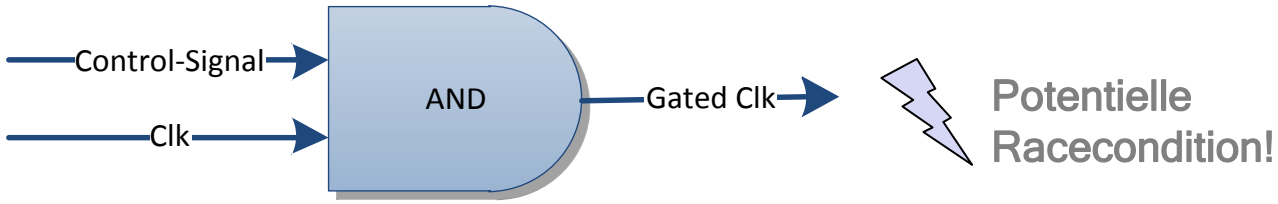
	0	1	2	3	4	5	6	7	8	9	10
T0	**5**	4	4	4	**4**	3	3	**3**	**2**	**1**	5
T1	2	**2**	1	1	1	**1**	~~0~~	~~0~~	~~0~~	~~0~~	**2**
T2	2	2	**2**	1	1	1	**1**	~~0~~	~~0~~	~~0~~	2
T3	1	1	1	**1**	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	~~0~~	1

03 Entwurf (6) - Architektur (vereinfacht)

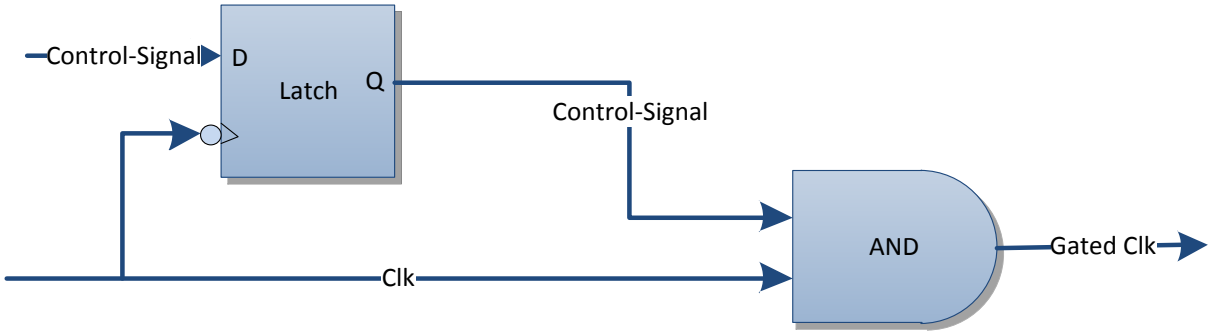


04 Energiesparmaßnahmen (1)

Clock-Gating (Latch-free)

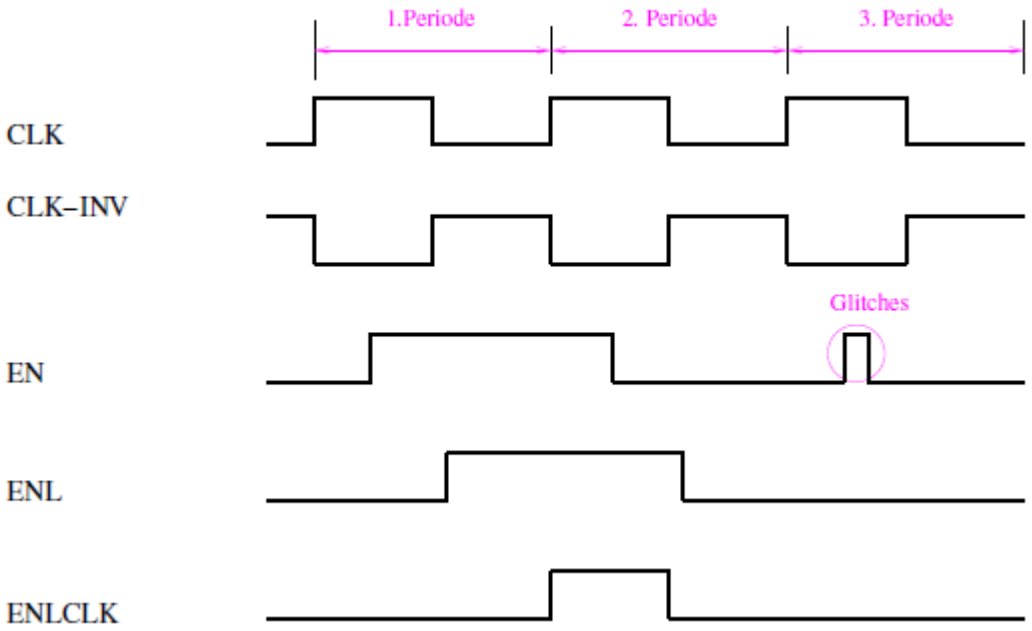


Clock-Gating (Latch-based)



04 Energiesparmaßnahmen (2)

Clock Gating mit T-LAT Zelle



04 Energiesparmaßnahmen (3)

Wo und Wozu werden Clock-Gating-Zellen eingesetzt?

- Bei 8 Threads mit je 16 Register sind pro Befehl und Takt höchstens 2 von 128 GPRs in Verwendung → Durchschnittlich1%.
- → Jedes General Purpose Register bekommt eine eigene Clock-Gating-Zelle
- Reduzierung des Stromverbrauchs sowie Belastung von Stützkapazitäten im Sleep- und Aktiven-Modus

05 Bewertung und Zusammenfassung (1)

Taktrate von 8 MHz reicht vollkommen aus, denn:

- Während sich die Kurbelwelle eines Bezinmotors bei 6000 U/min um 5° dreht sind mehr als 1000 Maschinenbefehle möglich.
- Busprotokolle LIN und CAN haben Datenübertragungen im Bereich von maximal 2 - 200 kBit. (etwa 1000 Maschinenbefehle pro Übertragung)

Multithreading

- Erlaubt Integration von mehreren Steuereinheiten auf einen Chip
- Reduziert Ressourcenverbrauch sowie eventuell nötigen Verkabelungsaufwand
- Erlaubt gezieltes Interrupthandling pro Thread

05 Bewertung und Zusammenfassung (2)

Ausgangslage

- Teure, dedizierte Hardware
- Konventionelle Mikroprozessoren von ATMEL sind für die Aufgaben bereits zu komplex und zu aufwändig

Bisherige Ergebnisse

- (Fast)-Vollständige, parametrierbare Multithreading Architektur mit passenden Befehlssatz
- Sparsamer und Einfacher Scheduling-Algorithmus
- Erste erfolgreiche Syntheseergebnisse
- Methoden zur Reduzierung des Stromverbrauchs mittels T-LAT-ZELLEN

05 Bewertung und Zusammenfassung (3)

Noch zu Bearbeiten

- Möglichkeiten zum Speichern des Programmcounters in Reg-File, bei ungünstiger Parametrierung
- Nutzen des Befehls LD_PC bewerten, ggf rausstreichen
- Clock-Gating-Problem zum Test auf dem FPGA
- Entwurf weiterer Testszenarien
- Eventuelle Optimierungen

Quellen

1 Chiviny Long-Marquardt:Clock Gating, Studienarbeit, Tübingen 2004

2 Frank Emnett, Mark Biegel. Power Reduction Through RTL Clock Gating, Automotive Integrated Electronics Corporation, SNUG San Jose 2000

3 HYPERTHREADING TECHNOLOGY IN THE NETBURST MICROARCHITECTURE, David Koufaty Deborah T. Marr, Hillsboro, OR, 2003

4 Wikipedia, Simultaneous multithreading, http://en.wikipedia.org/wiki/Simultaneous_multithreading

5 http://www.kfz-tech.de/DigSteuergeraet.htm

6 Autonews, http://www.autonews-123.de/wp-content/uploads/2009/09/BMW-Bordnetz-der-Zukunft.jpg (Bild)

7 http://www.csd-electronics.de/ (Bild)

Vielen Dank für die Aufmerksamkeit!

»Wissen schafft Brücken.«

A Auswertung (Ressourcen auf FPGA)

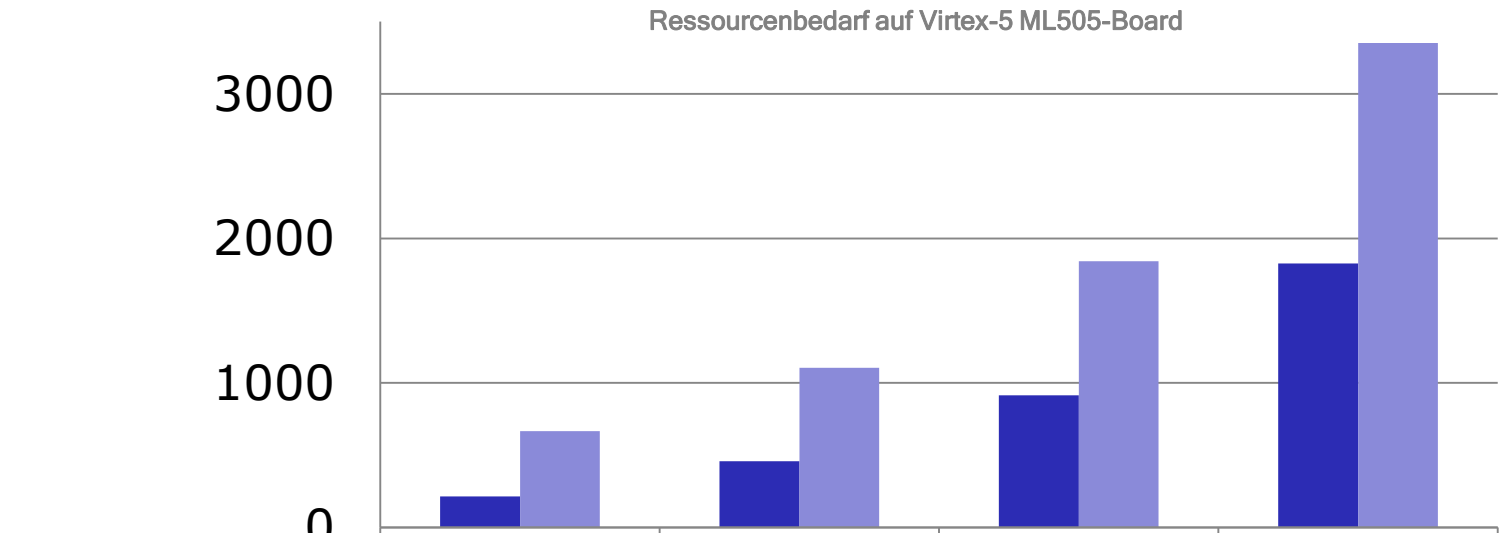
Ressourcenbedarf auf Virtex-5 ML505-Board

Threads	*1*	*2*	*4*	*8*
Register	215	458	914	1826
6-Input LUTs	666	1106	1841	3351
*Frequenz**	133 MHz	118 MHz	108 MHz	100 MHz

Synthese mit 16 Register pro Thread; Stacktiefe 3; 10 Bit Arithmetik

- Mehr als doppelt soviel LUTS wie Register
- Ohne externe Speicheranbindung

A Auswertung (Ressourcen auf FPGA)



	1	2	4	8
Register	215	458	914	1826
6-Input LUTs	666	1106	1841	3351

A Auswertung (Ressourcen auf 0,36µ Standardzellen)

Ressourcenverteilung bei Synthese mit 16 Register à 10 Bit pro Thread