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

Diplomverteidigung

Stephan Richter – stephan.richter@mailbox.tu-dresden.de

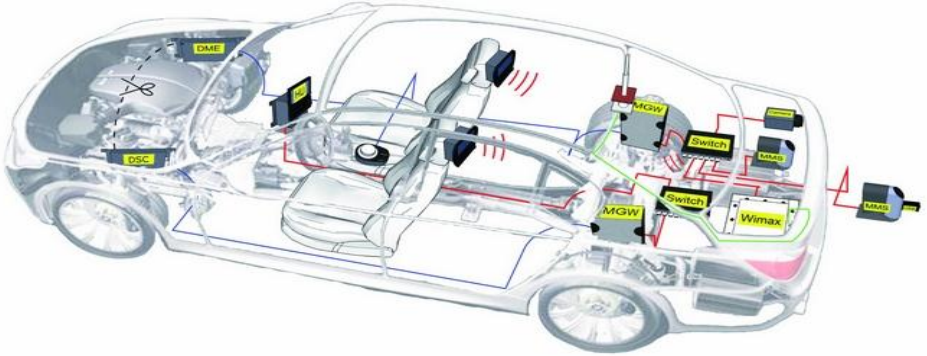
Dresden, 07.12.2011

Gliederung

1 Einführung - Anforderungen, Einsatzgebiet

- Einsatzgebiet: Automobilsektor

- Motorsteuerung
- Bus-Controller (LIN, CAN, SPI)
- Einparkhilfe (Park-distance-control)
- Fensterheber

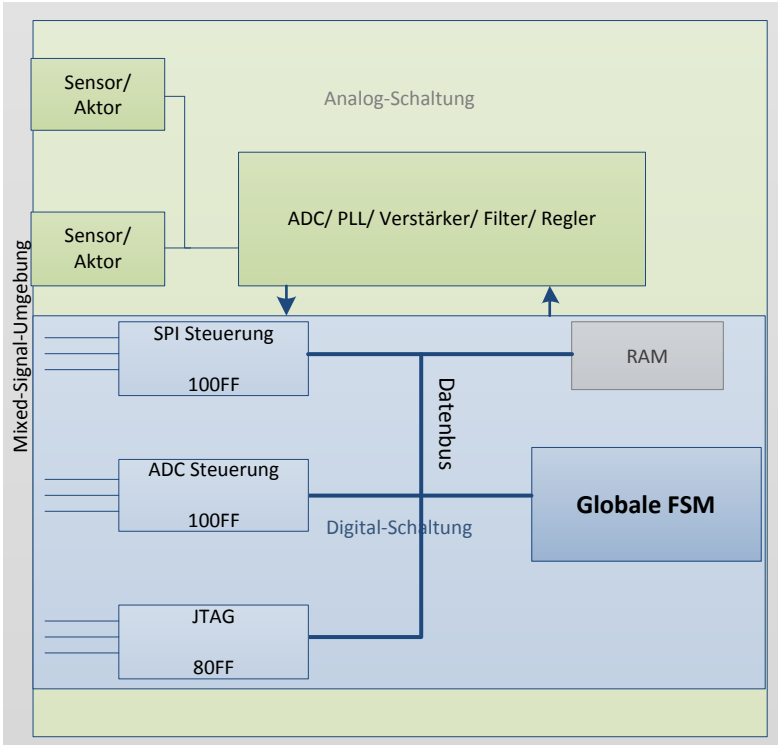


- Frei parametrierbare Hardware
- Energie- und flächeneffiziente Architektur

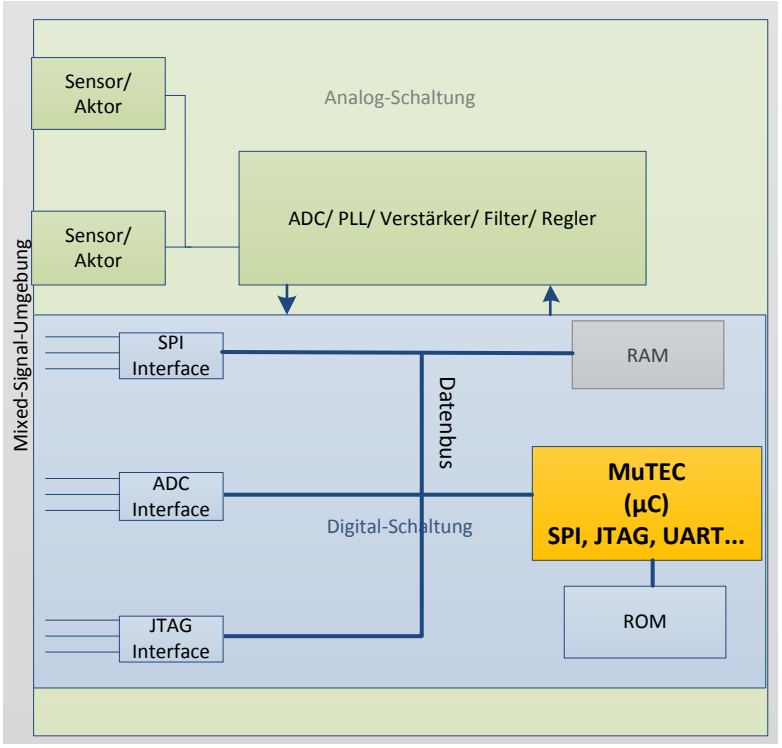
Quelle: Autonews, http://www.autonews-123.de/wp-content/uploads/2009/09/BMW-Bordnetz-der-Zukunft.jpg

Einführung - Mixed-Signal-Umgebung

Ohne Prozessor



Mit Prozessor



MuTEC: Multi-Threaded-Embedded-Controller

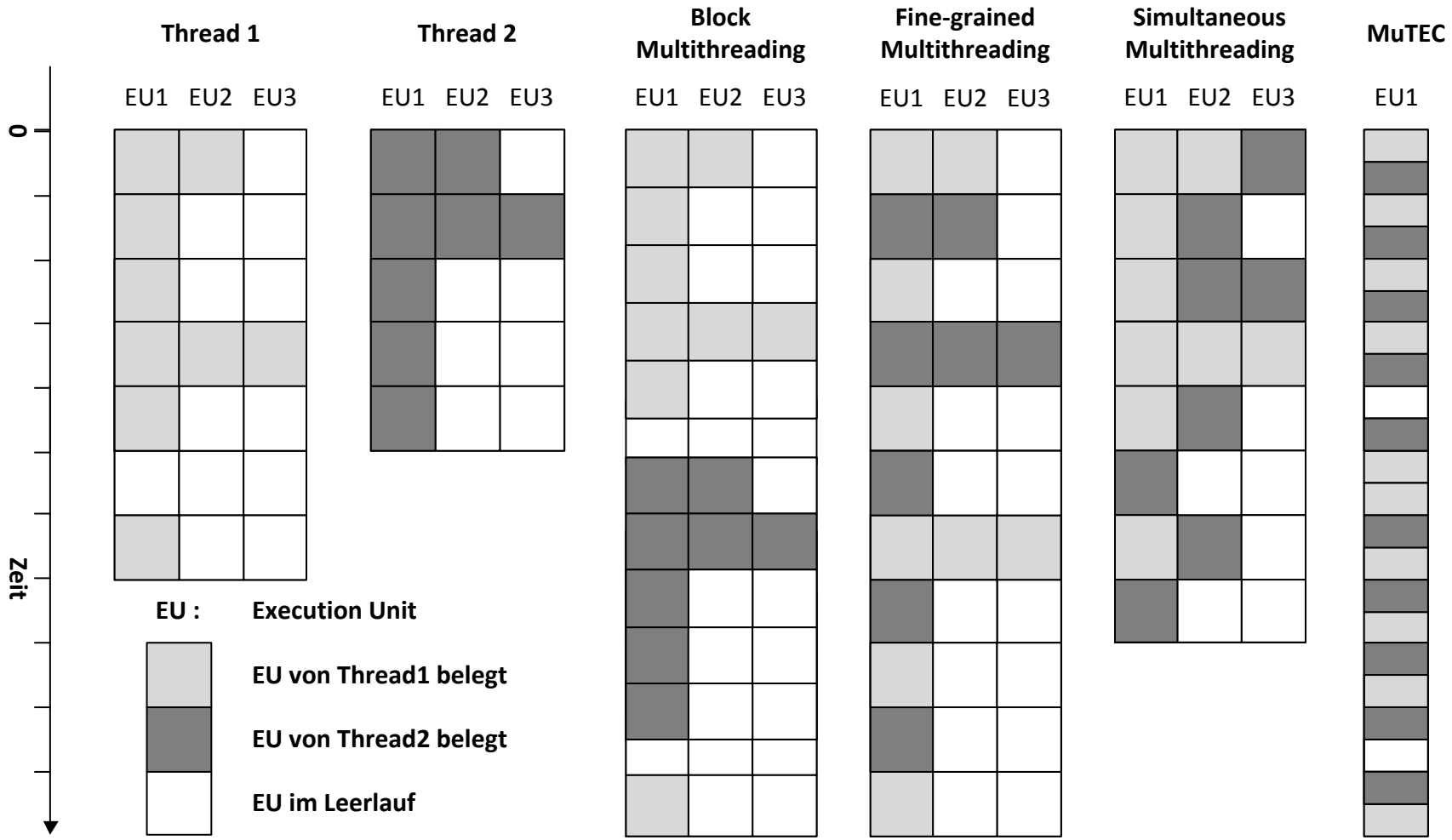
Einführung - Motivation

Wozu ein neuer Prozessor?

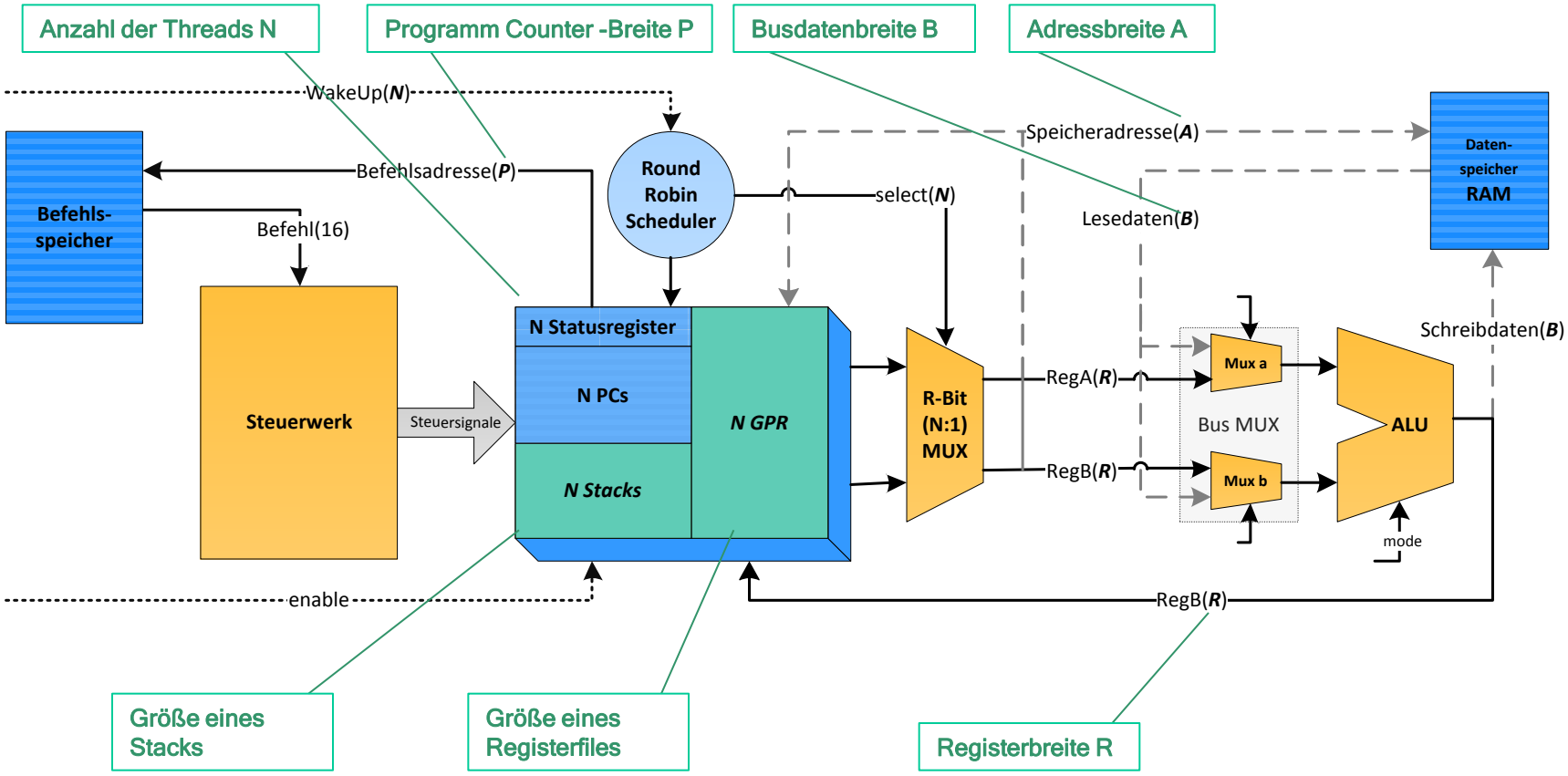
- Komplexität verfügbarer (Multi-Threading-) Prozessoren zu hoch, Integration zu aufwendig und kostenintensiv
- Geschwindigkeit nicht maßgebend
- Echtzeitfähigkeit und Parallelität
- Energie und Flächeneffizienz

- Große Bandbreite an Anwendungen
  - Eigener Befehlssatz
  - Parametrierbarkeit

Einführung - Hardwareseitiges Multithreading

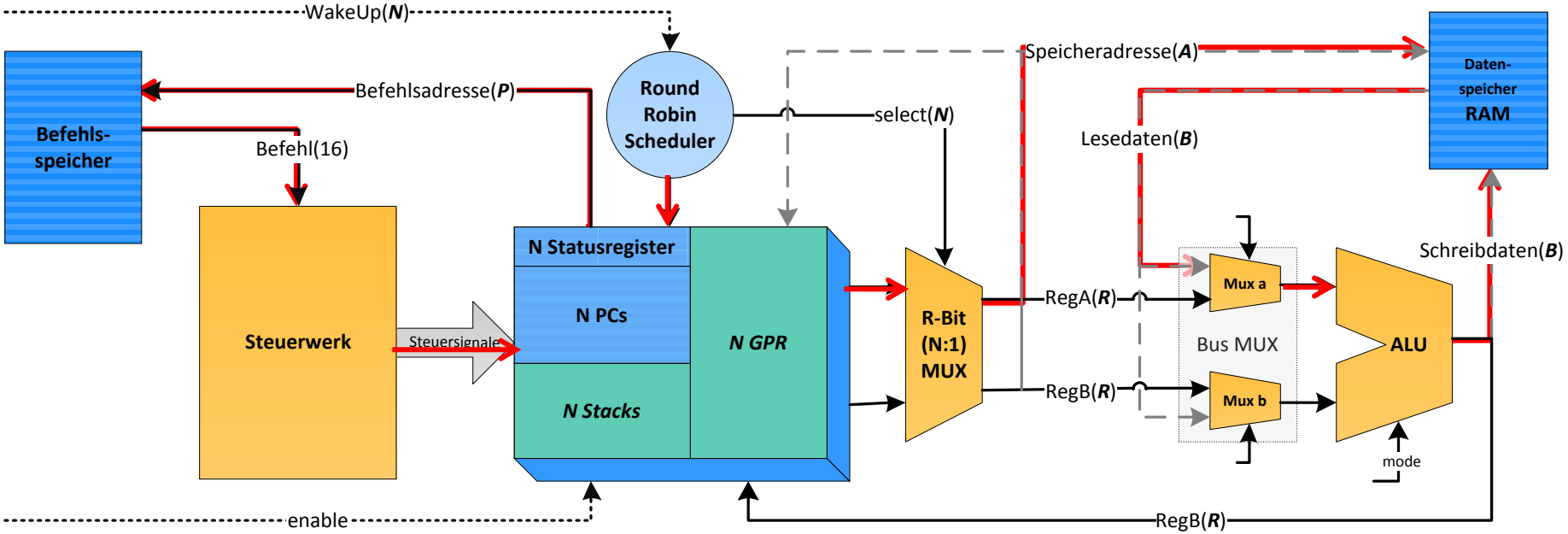


2 **Prozessorsystem** – Architektur, Parameter



Prozessorsystem – Architektur, kritischer Pfad

- Harvard Architektur
- Scheduler
- Kein Pipelining



Prozessorsystem – Befehlssatz Funktion

	NOP	Keine Operation
Sprung	**CALL**	Unterprogrammaufruf
Sprung	JMPR	Relativer Sprung
Sprung	BCZ (Branch)	Branch condition zero
Sprung	BCNZ	Branch condition not zero
Sprung	BCN	Branch condition negative
Sprung	BCNN	Branch condition not negative
Sprung	BCC	Branch condition carry
Sprung	BCNC	Branch condition not carry
Sprung	CALLI	Indirekter Unterprogrammaufruf
Sprung	JMPI	Indirekter Sprung
Sprung	LD_PC	Load Program Counter
Sprung	**RET**	Return (von Unterprogramm)
Scheduler	**SET_PRIO**	Setze eigene Priorität
Scheduler	**CLI**	Scheduler aus
Scheduler	**SEI**	Scheduler an

Math./Logische	ADD, SLL	Addieren
Math./Logische	ADC, ROL	Addieren m Carry
Math./Logische	SUB	Subtrahieren
Math./Logische	SBC	Subtrahieren m Carry
Math./Logische	AND	UND
Math./Logische	OR	ODER
Math./Logische	XOR	Exklusiv ODER
Math./Logische	CP	Vergleich
Math./Logische	LD	Lade Register zu Register
Math./Logische	ADDI	Addiere Konstante
Math./Logische	SUBI	Subtrahiere Konstante
Math./Logische	CPI	Vergleiche auf Konstante
Math./Logische	NOT	Negation (1-Kompl)
Math./Logische	CLR	Löschen
Math./Logische	SLR	Rechts schieben
Math./Logische	ROR	Rechts rotieren
Bitverab.	**LD ID**	Lade Konstante zu Register
Bitverab.	SETBIT	Setze Bit
Bitverab.	CLRBIT	Lösche Bit
Bitverab.	CHKBIT	Prüfe Bit (Carry)

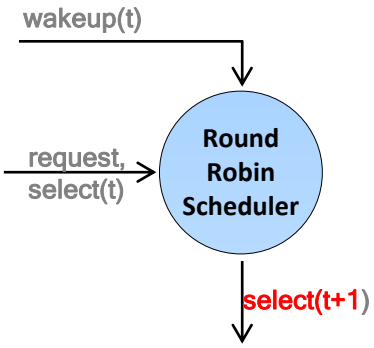
Prozessorsystem – Befehlssatz Codierung

	NOP	1111	1111	----	----
Sprung	**CALL**	1101	dddd	dddd	dddd
	JMPR	1110	vrrr	rrrr	rrrr
	BCZ (Branch)	1111	v000	rrrr	rrrr
	BCNZ	1111	v001	rrrr	rrrr
	BCN	1111	v010	rrrr	rrrr
	BCNN	1111	v011	rrrr	rrrr
	BCC	1111	v100	rrrr	rrrr
	BCNC	1111	v101	rrrr	rrrr
	CALLI	1111	0110	000b	bbbb
	JMPI	1111	0110	010b	bbbb
	LD_PC	1111	0110	100b	bbbb
	RET	1111	0110	11--	----
Scheduler	**SET_PRIO**	1111	0111	00pp	pppp
	CLI	1111	0111	01--	----
	SEI	1111	0111	10--	----

Math./Logische	ADD, SLL	0000	iaaa	aaib	bbbb
	ADC, ROL	0001	iaaa	aaib	bbbb
	SUB	0010	iaaa	aaib	bbbb
	SBC	0011	iaaa	aaib	bbbb
	AND	0100	iaaa	aaib	bbbb
	OR	0101	iaaa	aaib	bbbb
	XOR	0110	iaaa	aaib	bbbb
	CP	0111	iaaa	aaib	bbbb
	LD	1000	iaaa	aaib	bbbb
	ADDI	1001	dddd	ddib	bbbb
	SUBI	1010	dddd	ddib	bbbb
	CPI	1011	dddd	ddib	bbbb
	NOT	1111	1110	00ib	bbbb
	CLR	1111	1110	01ib	bbbb
	SLR	1111	1110	10ib	bbbb
	ROR	1111	1110	11ib	bbbb
Bitverab.	**LD ID**	1100	00dd	dddd	dddd
	SETBIT	1100	01ss	ssib	bbbb
	CLRBIT	1100	10ss	ssib	bbbb
	CHKBIT	1100	11ss	ssib	bbbb

Prozessorsystem – Scheduling

Round Robin Scheduler (Zeitscheibenverfahren)



Benötigtes Verhalten:

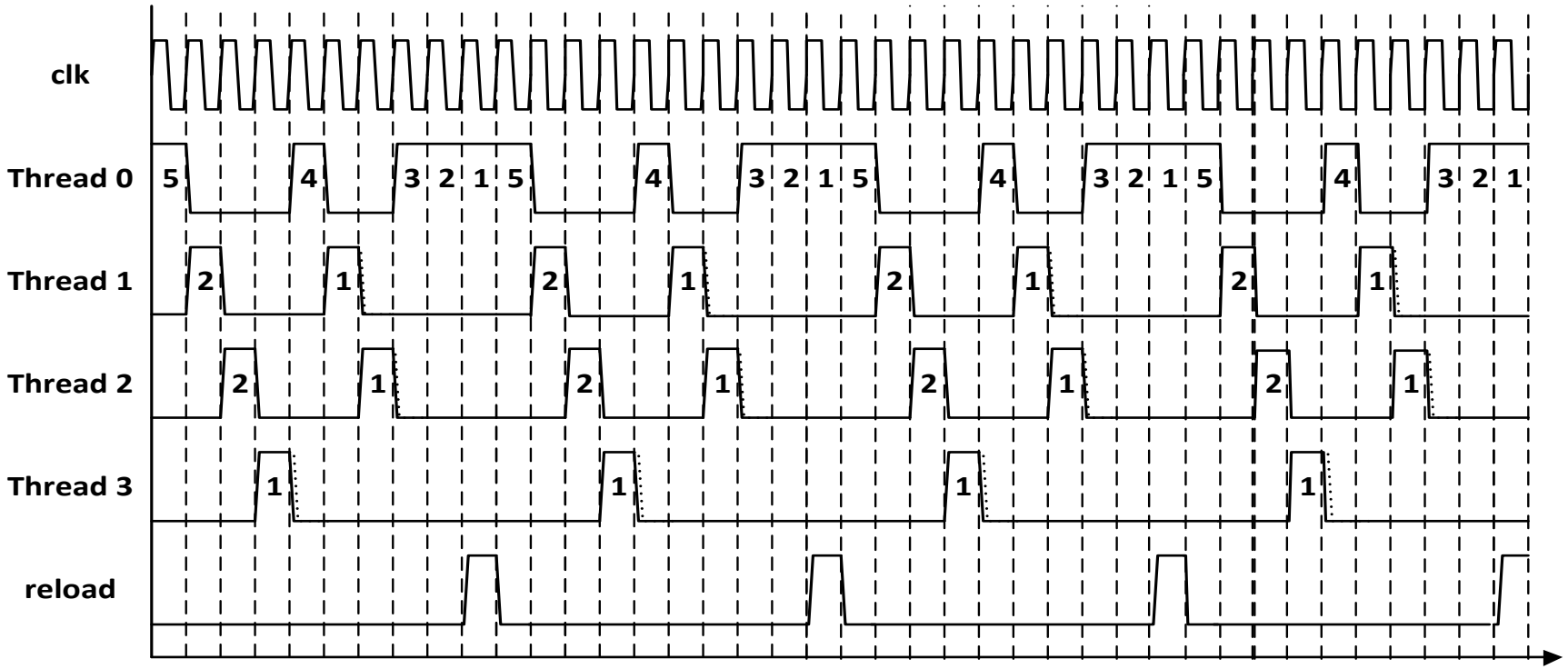
request(N, t) &	select(N, t) =>	select(N, t+1)
11…11	00…01	00…10
11…11	10…00	00…01

$$\boldsymbol{request_{left}(N,t)} \Leftarrow \overline{(\boldsymbol{select(N,t)} - \mathbf{1}) \cup \boldsymbol{select(N,t)}} \cap \boldsymbol{request(N,t)}$$

$$\boldsymbol{select(N,t+1)} \Leftarrow \begin{cases} \overline{(\boldsymbol{request_{left}(N,t)} + \mathbf{1})} \cap \boldsymbol{request_{left}(N,t)} & | \ \boldsymbol{request_{left}(N,t)} \neq \mathbf{0}, \\ \left(\overline{\boldsymbol{request(N,t)}} + \mathbf{1}\right) \cap \boldsymbol{request(N,t)} & | \ \boldsymbol{request_{left}(N,t)} = \mathbf{0} \end{cases}$$

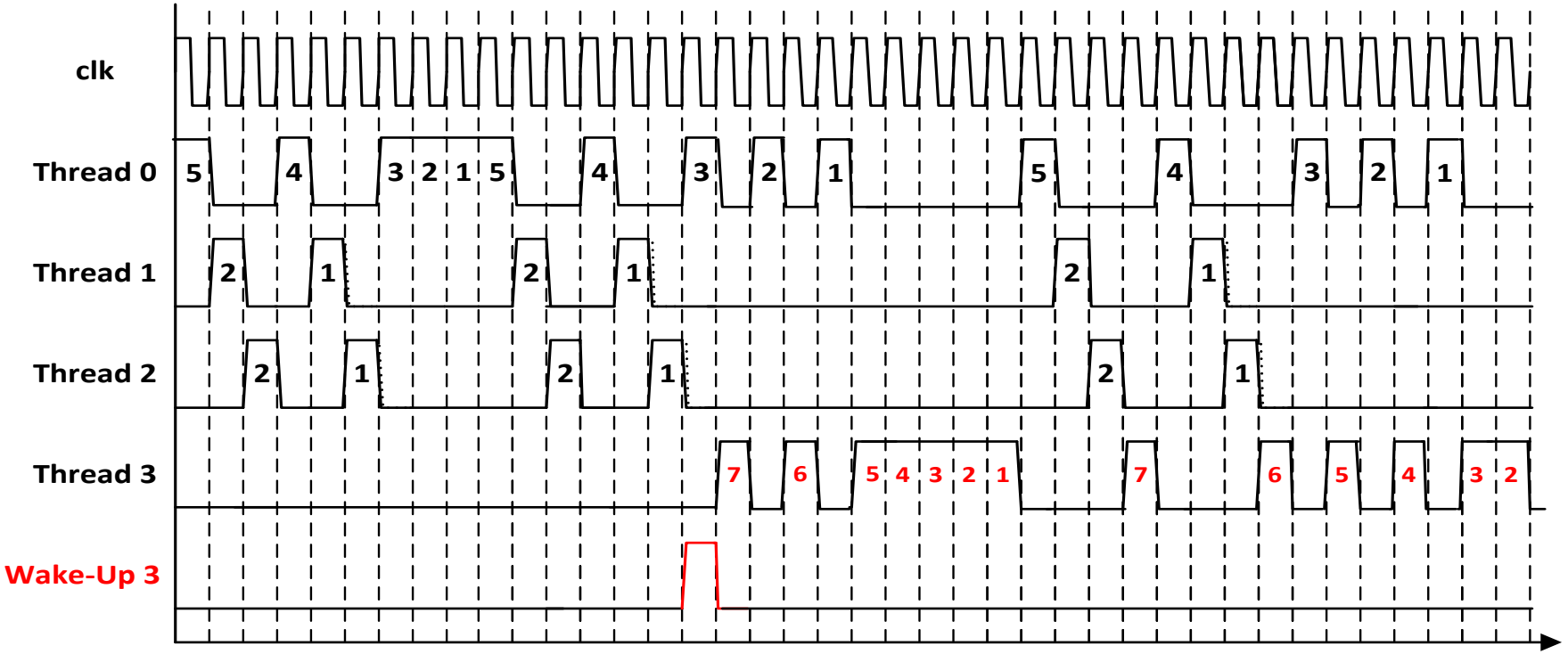
Prozessorsystem – Scheduling

Beispiel: 4 Threads mit Prioritäten (5,2,2,1) über vier Perioden

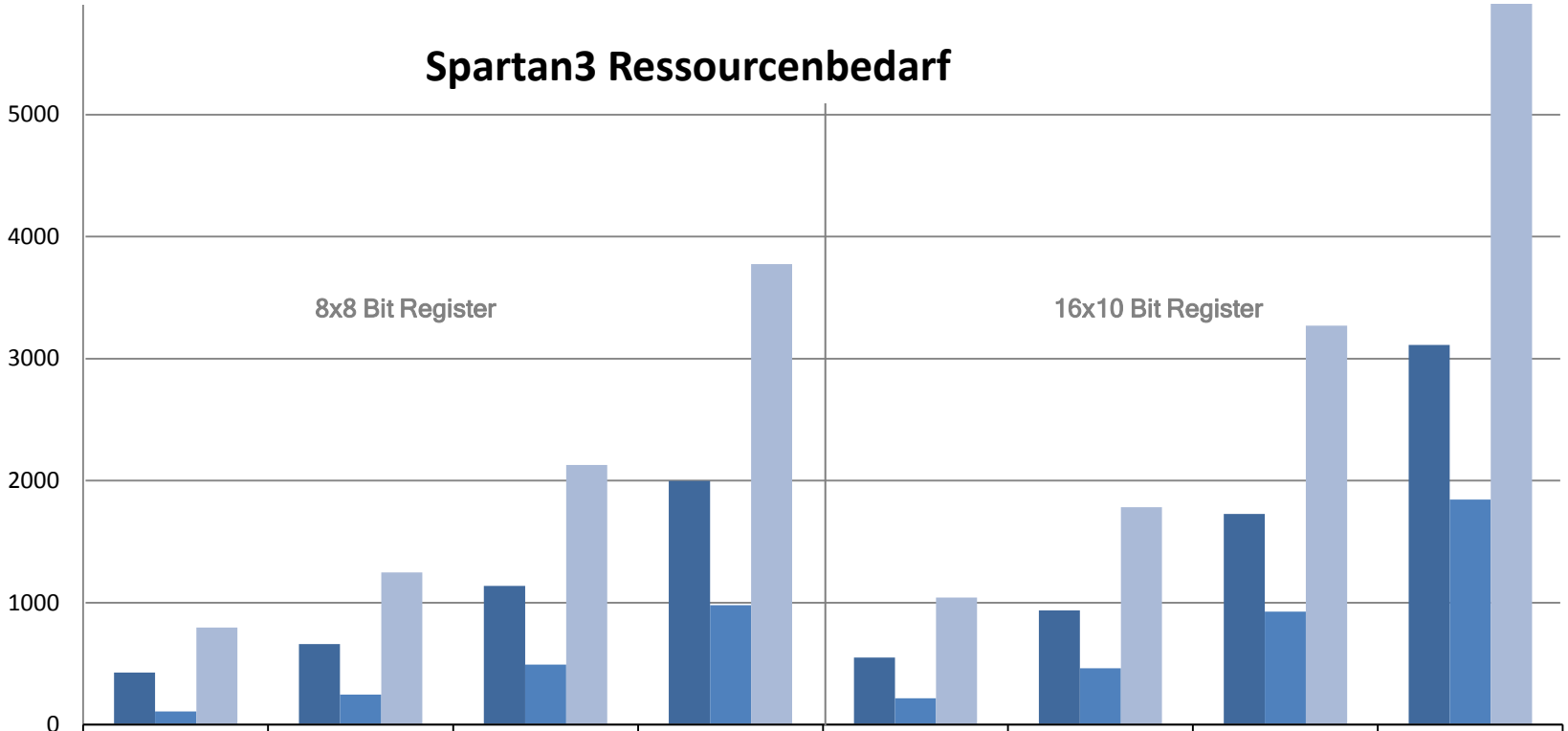


Prozessorsystem – Wake-Up

Beispiel: Thread 3 wird „aufgeweckt“



3 Realisierungen – FPGA



	1Th	2Th	4Th	8Th	1Th	2Th	4Th	8Th
Slices	428	661	1138	1999	550	937	1727	3111
Slice Flip-Flops	107	247	492	980	215	462	925	1845
4-Input LUTs	795	1248	2128	3775	1042	1783	3270	5931

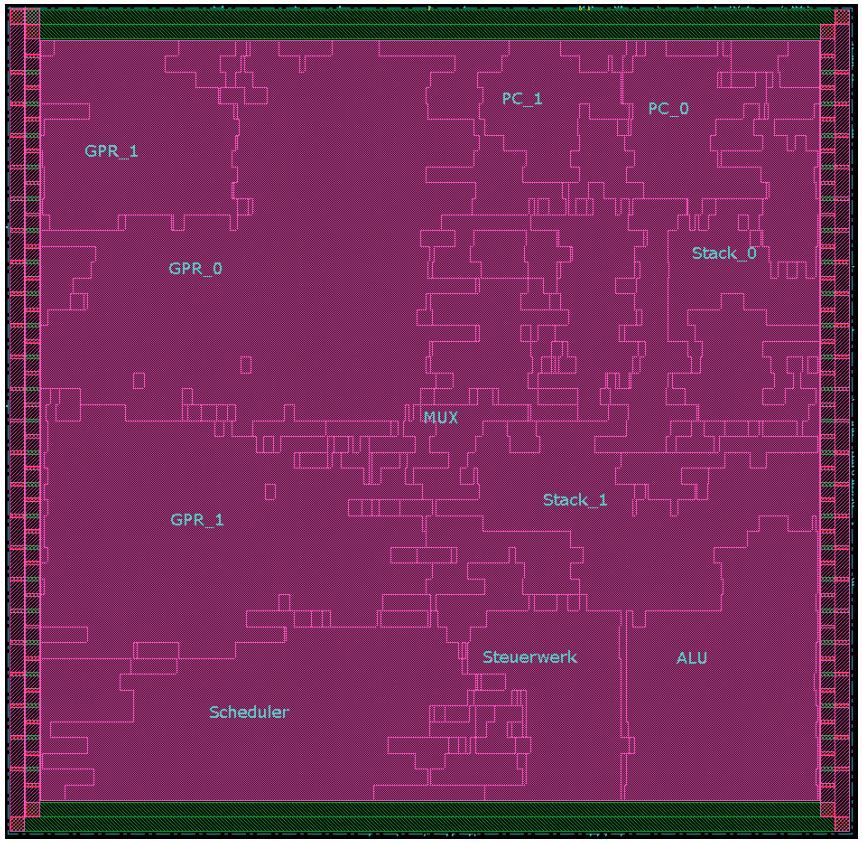
Realisierungen – ELMOS 0,35µ-Standardzellen

Place & Route für 8 Konfigurationen:

- N: 1, 2, 4, 8 Threads
- Mit und ohne Clock-Gating

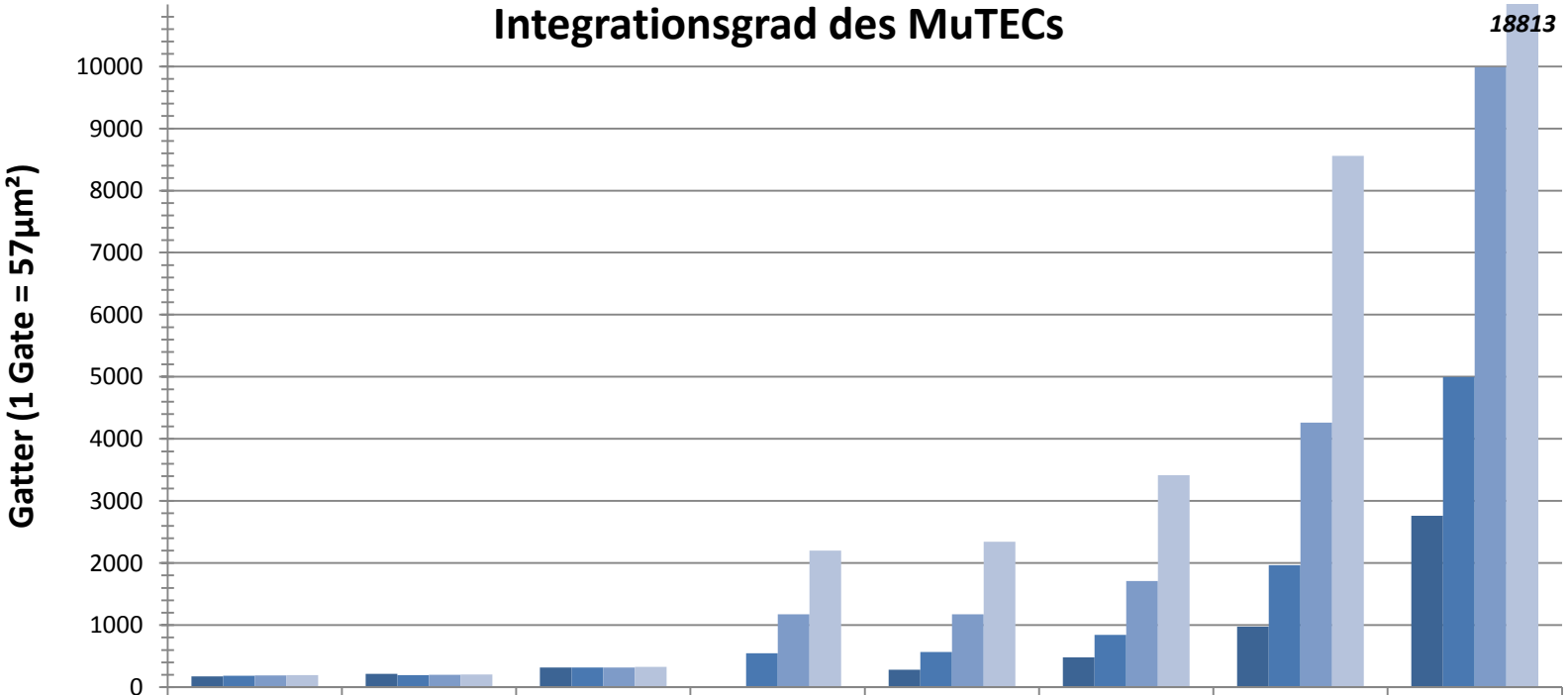
Weitere Parameter:

- R: 8x8 Bit Register / Thread
- 2 Stackeinträge / Thread
- P: 12 Bit PC
- B: 8 Bit Datenbus
- A: 8 Bit Adressbreite



N=2, P=12, B=8, R=8, A=8,

Realisierungen – ELMOS 0,35µ-Standardzellen

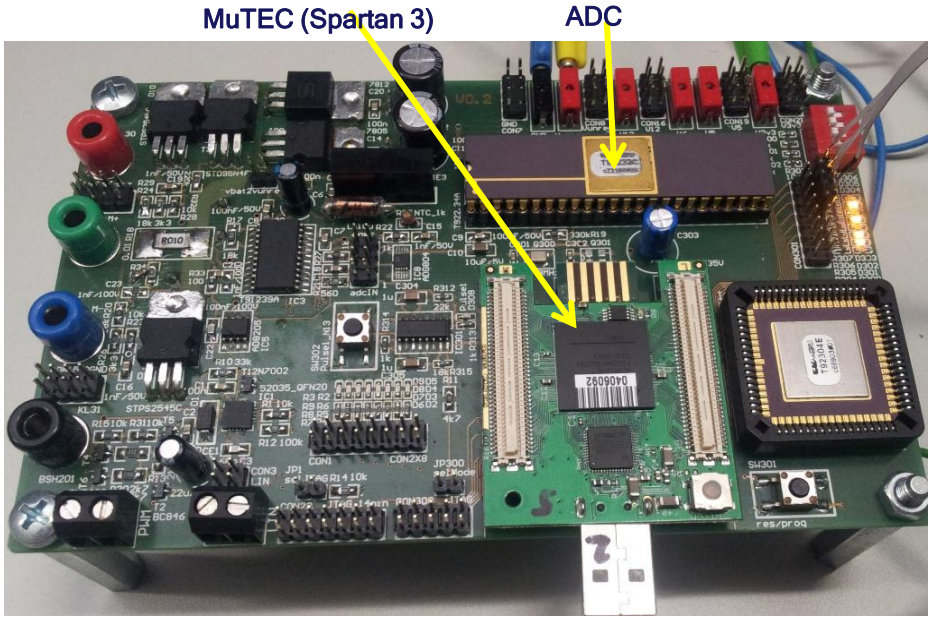


	Bus Mux	Steuerwerk	ALU	Scheduler	Stack	PC	GPR	MuTEC
1 Thread	174	215	316	0	282	480	976	2760
2 Threads	183	197	316	545	568	844	1964	4997
4 Threads	191	199	315	1172	1172	1712	4260	9990
8 Threads	197	206	325	2198	2344	3416	8560	18813

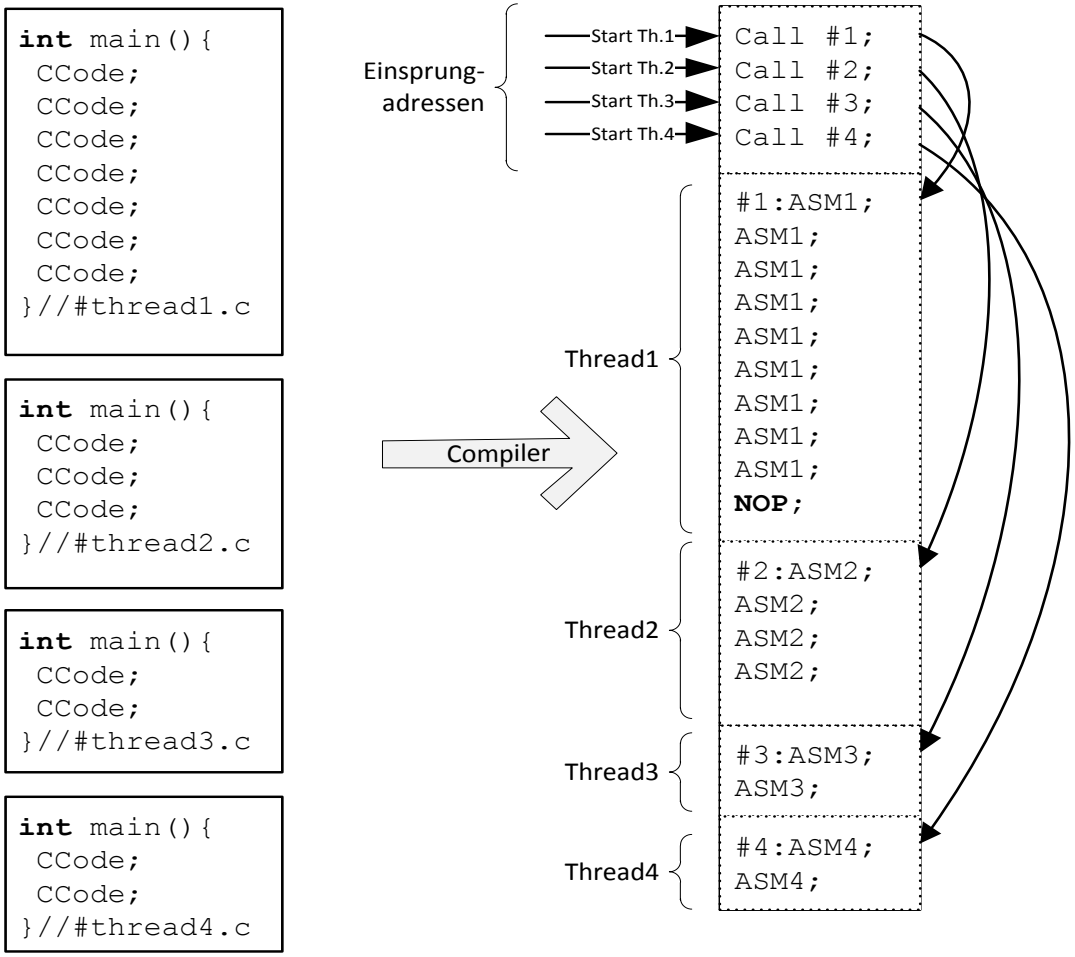
4 Verifikationsmethoden

1. Verifizierung einzelner Komponenten (97,1% Abdeckung)
2. Simulation von „selbst-testenden-C-Programmen“ (links)
3. FPGA-Validierung mit ADC-Steuerung

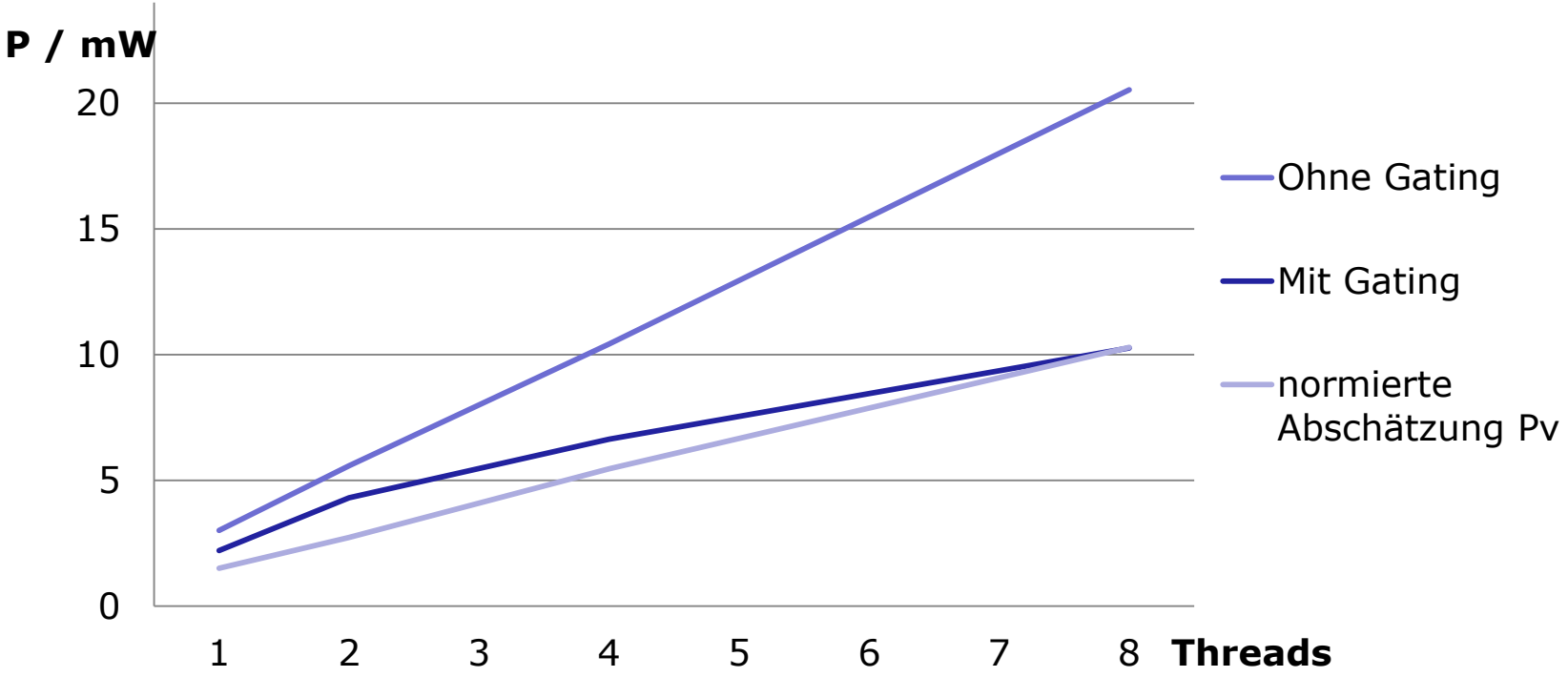
```
void main(){
  …
  result = mul_c(16,16);

  if ((256) != result)
    return;  // Fehler
  if (256 > result)
    return;  // Fehler
  if (256 < result)
    return;  // Fehler
  if (255 >= result)
    return;  // Fehler
  if (257 <= result)
    return;  // Fehler
  …
  return; // Programmende
}
```



Verifikationsmethoden – Kompilierung



5 Leistungsaufnahme



$V_{DD} = 3{,}6\,V$

$I_{DD} = \frac{19\mu A}{MHz * 1000\ Gatter}$

$P_v \approx \frac{N_{Gate} * V_{DD} * \overline{I_{DD}} * f_{clk}}{1000 Gates * MHz}$

6 Zusammenfassung

Ergebnisse

- Funktionstüchtige Validierungsumgebung auf einem Spartan 3 FPGA mit maximal 31 MHz
- 0,35µm CMOS-Standardzellen Realisierung bis 8 MHz
- Normierte Stromaufnahme bei 3,6V: $I_{DD} = \frac{19\mu A}{MHz * 1000\ Gatter}$

Ausblick

- Befehlssatzerweiterung
- Optimierungen (kritischer Pfad und Flächenbedarf)
- Compileroptimierung
- Praxisnahe Realisierungen:
  - Glühkerzensteuerung
  - DC-Motoransteuerung

Vielen Dank für die Aufmerksamkeit!

»Wissen schafft Brücken.«