Eingebettete Prozessorarchitekturen

7. Rekonfigurierbare Prozessoren

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

Wintersemester 2010/2011

Material und Literatur

- Auszüge aus Trainingsmaterial der Fa. Stretch
  - Insbesondere alle Zeichnungen und technischen Daten
- Material ist vertraulich
  - Nur für die Lehre zur Verfügung gestellt
  - Darf nicht weiterverbreitet werden

Bisheriger Stand
Eingebettete VLIW-Prozessoren

- Einfache Hardware-Strukturen für ILP
- Verlagerung des Aufwands auf Compiler
- *Grob* an Erfordernisse anpassbar
  - Anzahl an allgemeinen Ausführungseinheiten (ALU, MUL, ...)
  - Anzahl an Registerfeldern
- Vor der Chip-Fertigung

Bisheriger Stand
Konfigurierbare Prozessoren

- Sehr einfacher Basis-Prozessor
- Einfache sequentielle Ausführung
- *Nicht* optimiert auf ILP
- Fein an Erfordernisse anpassbar
  - Konfigurierbare spezielle Recheneinheiten
    - MAC, Vector, ...
  - Erweiterbarkeit um eigene Instruktionen
    - Haben eigenen Speicherzugriff
    - I.d.R. nur wenige Takte lang, kein komplexen Automaten
    - VLIW-artiges Modell möglich
- Anpassung vor der Chip-Fertigung

Diskussion VLIW und konfigurierbare CPUs

- Abstimmen auf unterschiedliche Anwendungen erfordert Chip-Fertigung
- Nur bei sehr großen Stückzahlen wirtschaftlich
- Änderungen nur beschränkt möglich
  - Nur soweit in ursprünglichem Aufbau vorgesehen
- Änderbarkeit ist aber praktisch relevant
  - Viele “Standards” ändern sich leicht vor Festlegung
  - Nachrüsten von Features der Konkurrenz
  - Korrigieren von Fehlern (gibt es auch in Hardware!)

➥Mehr Flexibilität wünschenswert

Rekonfigurierbare Prozessoren
Idee

- Änderbarkeit des Prozesors auch nach der Fertigung
  - Ohne photochemische Prozesse
  - Nennt sich Rekonfigurierbarkeit
- Problem
  - Rekonfigurierbare Strukturen kosten mehr Platz als feste
  - . . . daher nicht alles rekonfigurierbar halten
    - Wäre der FPGA-Ansatz
  - Stattdessen gezielt rekonfigurierbare Bereiche einbauen
    - Hier verallgemeinert Reconfigurable Area (RA) genannt

Flächenaufwand der Rekonfiguration

- Rekonfigurierbare Funktionen größer als feste, aber . . .
- Gleiche RA kann unterschiedliche Funktionen bereitstellen
- Wenn Funktionen nicht gleichzeitig gebraucht werden
- Wiederverwenden der teuren RA möglich
- Kann sogar billiger sein, als alle evtl. gebrauchten Funktionen als feste Logik zu realisieren!
  - Beispiel: Digitaler Rundfunk (verschiedene Empfangsstandards)

Geschwindigkeit der Rekonfiguration

- Wie schnell ist die Rekonfiguration möglich?
- Sekunden . . . Statische Rekonfiguration
  - Ändern der konfigurierten Funktion lohnt sich nur selten (zu langsam!)
  - Umschalten von Betriebsmodi
    - H.264 oder VC-1 Video Decoder, Selbsttestmodus
- Millisekunden . . .
  - Gelegentliche Änderung akzeptabel
  - Anpassung an *verschiedene* Anwendungen bei jeweiligem Programmstart
    - Spezielle Beschleuniger z.B. für Bildverarbeitung, Crypto, . . .
- Mikrosekunden . . . Dynamische Rekonfiguration
  - Änderungen auch *während* der laufenden Anwendung praktikabel
  - Anpassung an verschiedene *Teile* der Anwendungen
    - Beschleuniger für Unterfunktionen zum Filtern, Quantisieren, Komprimieren, . . .

Architekturideen

Embedded Systems & Applications

A. Koch

Rekonfigurierbarer Prozessor koppelt

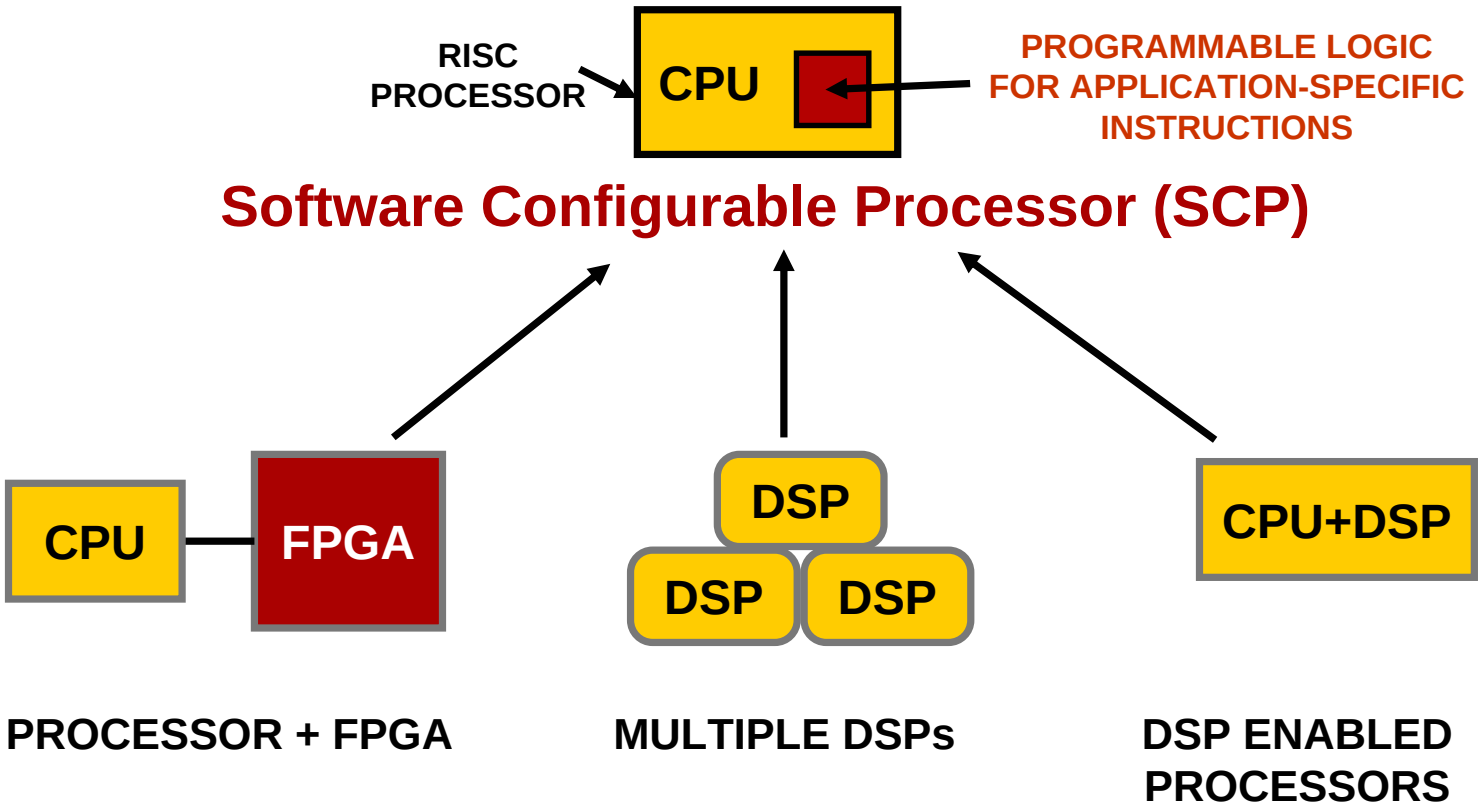
- Feste Standard-Prozessor-Pipeline der CPU
- RA eingebunden direkt in Pipeline als Funktionseinheit(en)
- Enge Kopplung, RA kann aber CPU-Pipeline verlangsamen

Kontrast: Adaptiver Computer (später) kombiniert

- Festen Standard-Prozessor
- RA angebunden an Prozessorbus auf Cache-Ebene
- *Nicht* mehr nur in Pipeline
- Lose Kopplung, RA und CPU agieren gleichberechtigt parallel
  - heterogener Multi-Core, manchmal sogar MPSoC

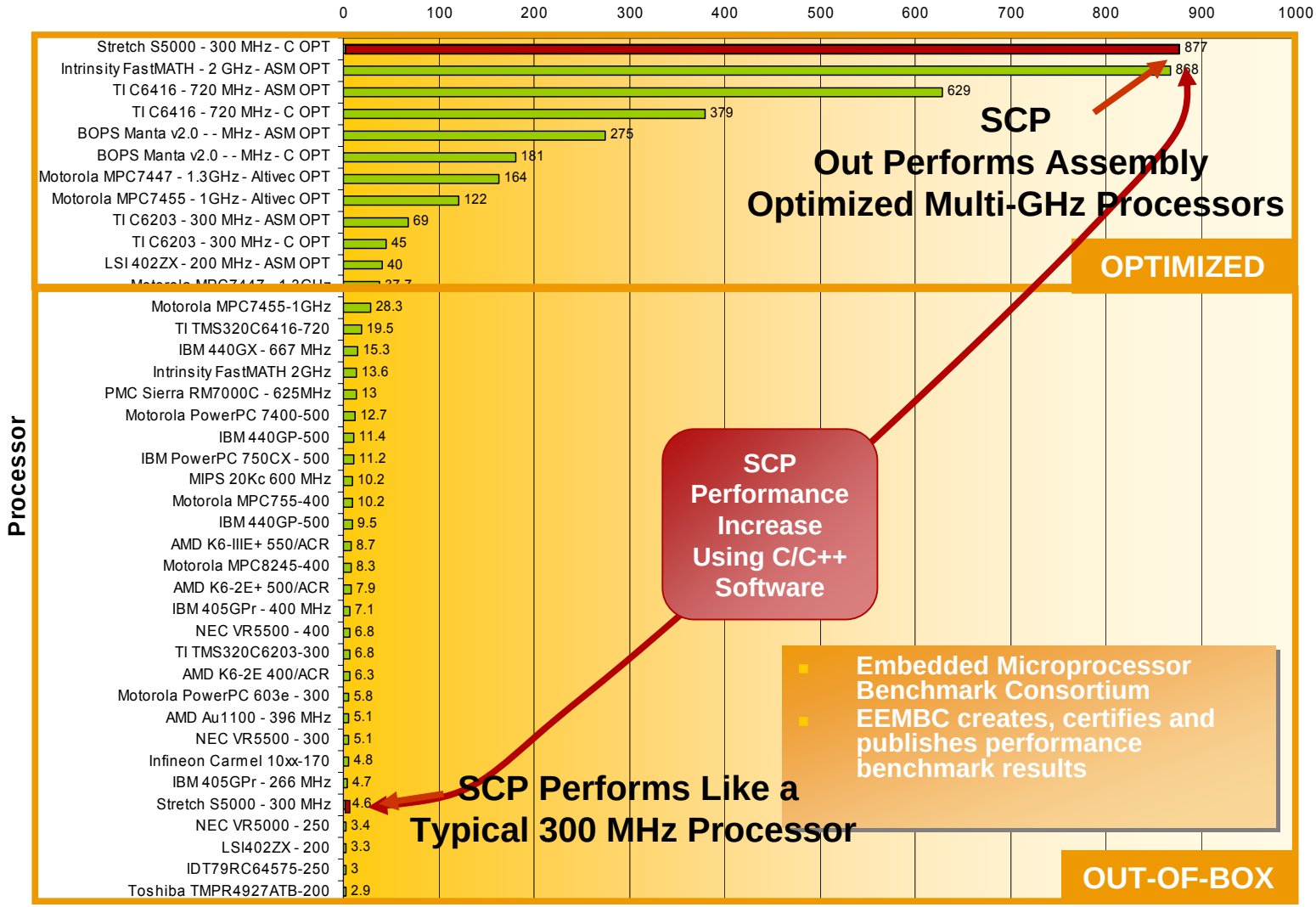
Rekonfigurierbarer Prozessor Stretch S5000

A. Koch



RA eingebunden in Pipeline

Gewinn an Rechenleistung



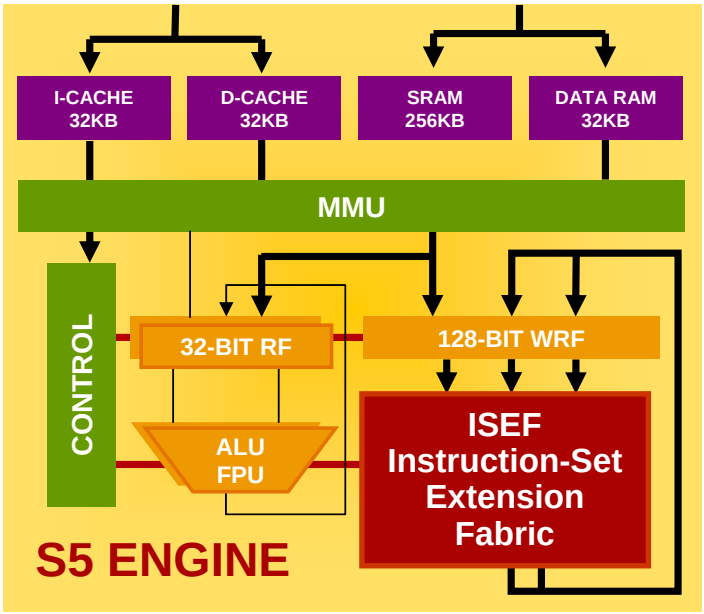
A. Koch

Ähnlich *konfigurierbarem* Prozessor, nun aber Funktion änderbar

Kernarchitektur *S5 Engine*

Mix aus festen und rekonfigurierbaren Komponenten

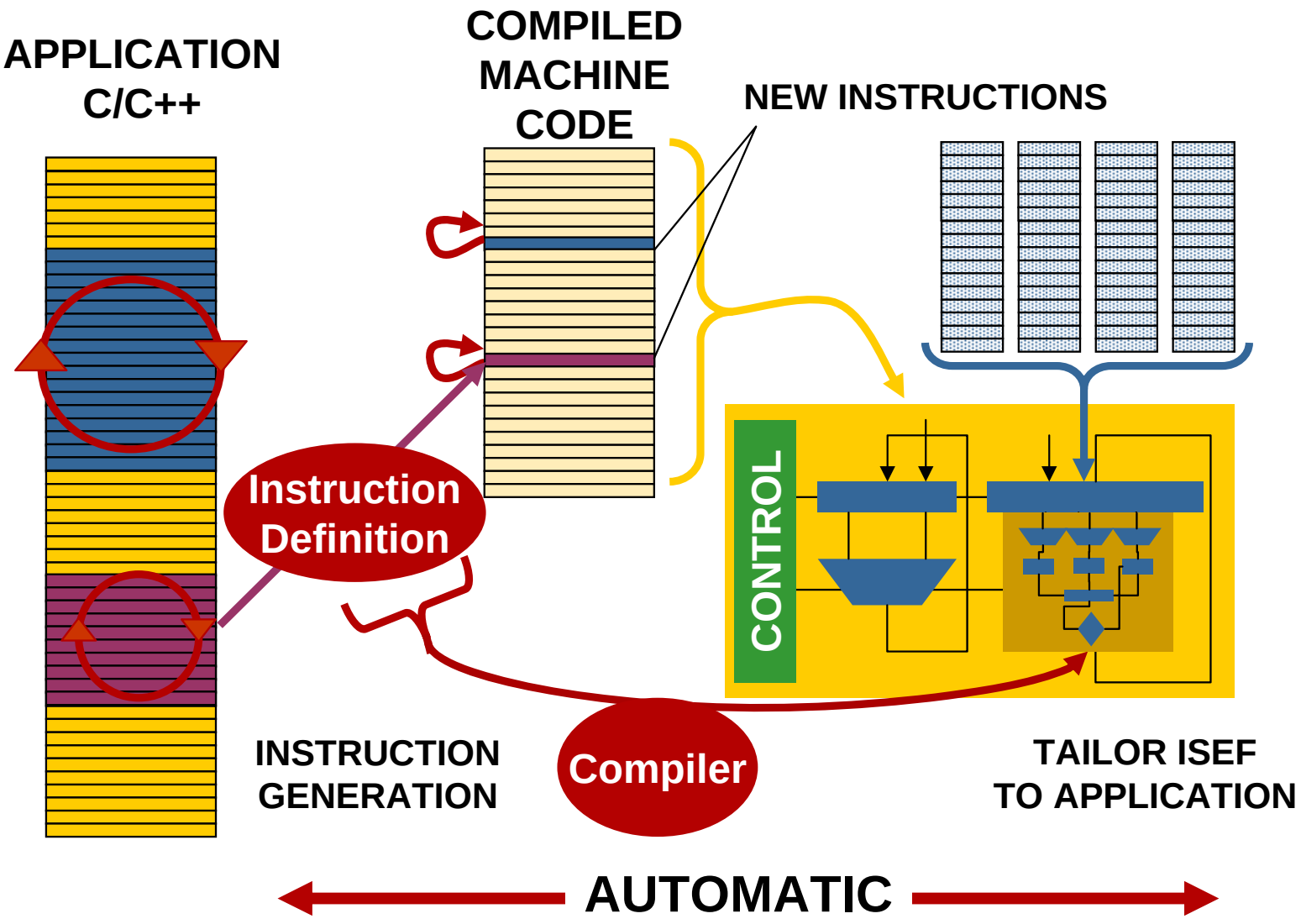
A. Koch



RISC Processor

Programmierfluß

Hier: C-artiges Stretch-C statt Verilog-artiges TIE



A. Koch

Vergleich Tensilica / Stretch

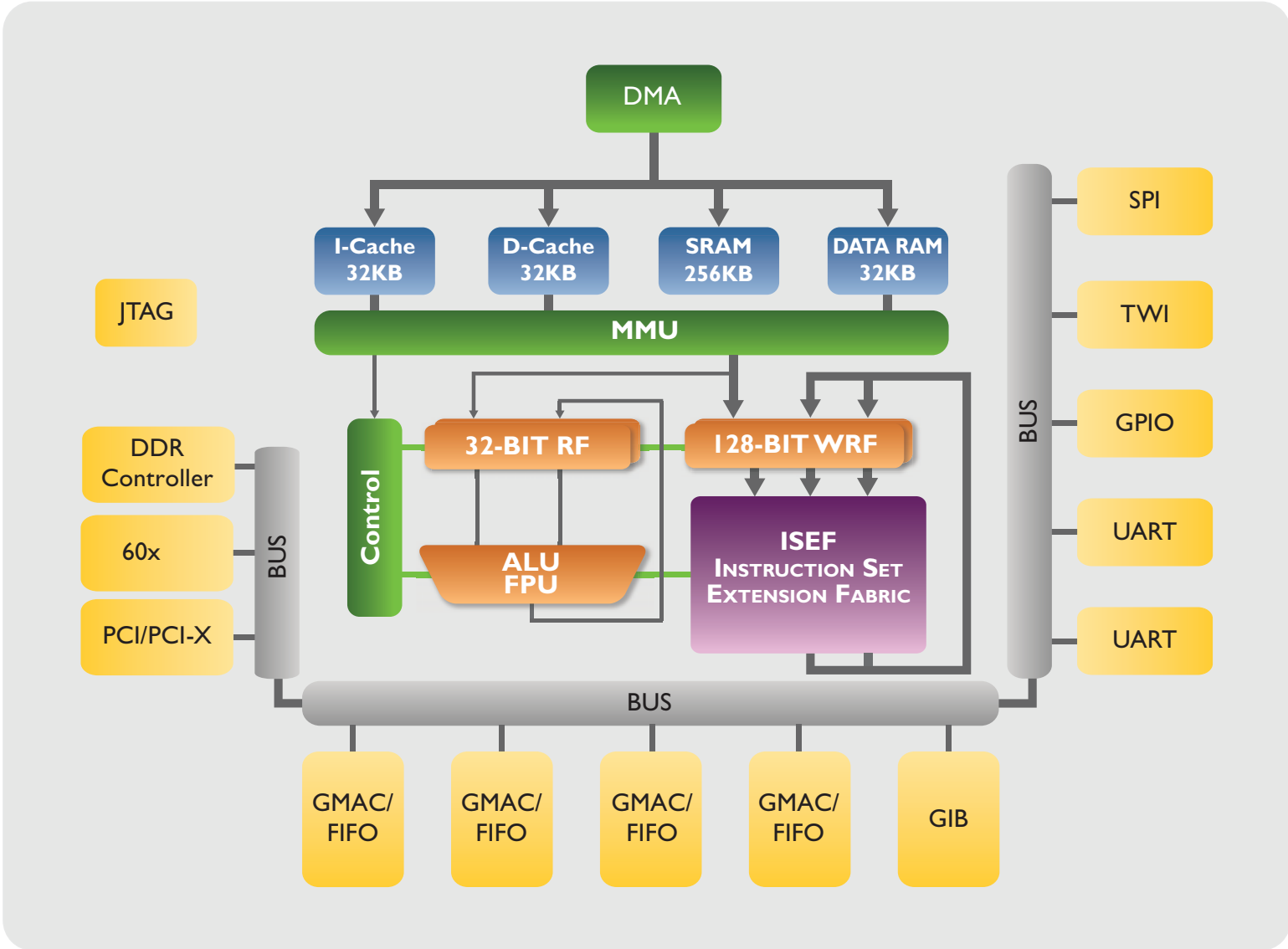
- Funktionalität wäre auch mit Tensilica-Technologie realisierbar
- Genau so wurde es gemacht
  - Ausnahme: RA in Form des ISEF
  - Stretch-eigener Hardware-Block
- Aber:
- Tensilica Werkzeuge produzieren Teilschaltung für größeren Chip
  - Sogenannter Intellectual Property-Block (IP Block)
  - Kunden sind für Chip-Fertigung selbst verantwortlich
- Stretch bietet fertige Chips an
  - Verschiedene Hardware um S5 Engine herum

Stretch S5620 Prozessor

S5 Engine mit Peripherie

Embedded Systems & Applications

A. Koch



Diskussion

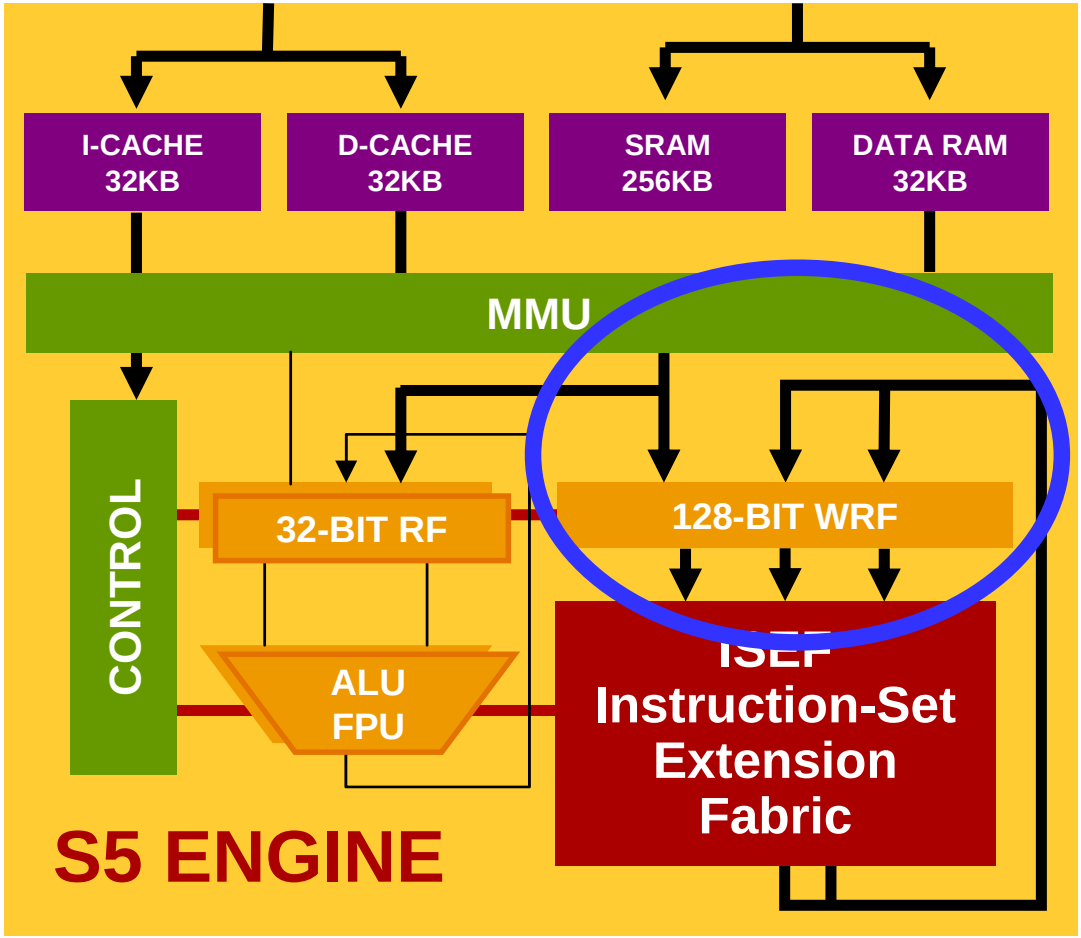
- Noch mehr feste Bestandteile
- Realisierung von Standardschnittstellen
  - Würde in RA viel Fläche benötigen
- Kann aber im Einzelfall ineffizient werden
  - Vielleicht braucht meine Anwendung gar kein Ethernet?
  - Die Fläche musste ich trotzdem bezahlen

➥Sorgfältiges Abwägen bei Systemkonzeption erforderlich!

Datenaustausch zwischen RA und festen Komponenten

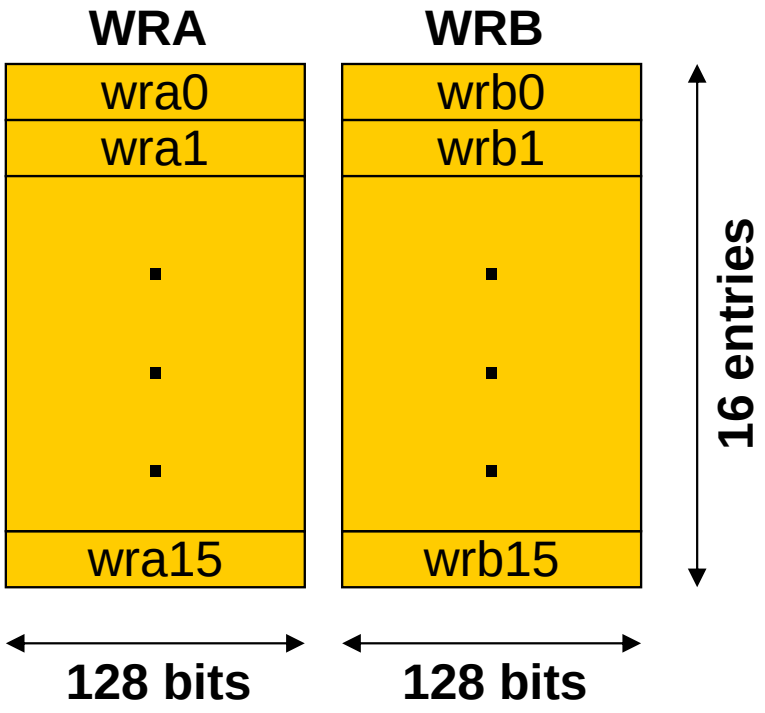
Nur über Wide-Register File (WRF)

A. Koch



- ISEF hat keine Speicherschnittstelle
- Muß LSU des festen Prozessorkerns benutzen

Aufbau des WRF



- Zwei parallele Bänke
- 128b breite Einträge
  - Beliebig organisiert!
  - 4x 32b Integer
  - 5x RGB Pixel von je 3 Bytes
  - 12x 10b Datenworte
- Benutzung aus C
  - Analog zu Tensilica-Registerfeld
  - `WRA x; WRB y;`
  - `WR z;`
    - Compiler nimmt Zuordnung vor

Beispiel für Benutzung des WRF

Kopiere Feld aus 4 Integers

A. Koch

```
/* assume a and b are  */
/* 16-byte aligned     */
copy4int_implicit(int *a, int *b)
{
    WR t;
    WR *pa = (WR *) a;
    WR *pb = (WR *) b;

    t = pb[0]; /* load to t */
    pa[0] = t; /* store from t */
    /* same as pa[0] = pb[0]; */
}
```

C

```
# 2 cycles to move 16 bytes
entry    a1,32
wral128i wra0,a3,0
wras128i wra0,a2,0
retw.n
```

Stretch Assembler

Ähnlich zu Tensilica

Neue Adressierungsmodi: Postinkrement

Kopiere Feld aus N Integers


```c
/* copy N integers */
copyNint_IU(int *a, int b, int n)
{
    int i;
    WR t;
    WR *pa = (WR *) a;
    WR *pb = (WR *) b;

    for (i = 0; i < n/4; i++) {
        WRL128IU(&t, &pb, 4*4);
        WRS128IU(t, &pa, 4*4);
    }
}
```

C

```
entry       a1,32
mov.n       a9,a2
mov.n       a8,a3
addi.n      a10,a4,3
movgez      a10,a4,a4
srai        a10,a10,2
loopgtz     a10,.LBB28_copyNint_IU
  wral128iu   wra0,a8,16
  wras128iu   wra0,a9,16
.LBB28_copyNint_IU:
retw.n
```

Stretch Assembler

- Erhöht Adressregister um gegebene Konstante
  - Hier: Jedesmal um 16 Bytes
- Wie bei Tensilica: Pseudo-Funktionen aus C

Übersicht über ausgerichtete Load/Store-Anweisungen

Bank	Operation	Size (bits)	Mode
WR WRA WRB	L (load) S (store)	128 64 32 16 8	I (immediate offset) IU (immediate post increment) X (register offset) XU (register post increment) CU (circular buffer) RU (bit-reverse)

- Beispiel: **`WRAL64IU(target, base, 8)`**
- Daten müssen im Speicher der Zugriffsgröße entsprechend ausgerichtet sein
  - Bei n-bit Zugriff an n-bit Grenze
  - Beispiel: Bei 128b Zugriff muß Adresse durch 128b/8b = 16 Bytes teilbar sein
  - In C durch Attributieren der Deklaration

```
int A[1024] __attribute__ ((aligned(16)));
```

Erstes Anwendungsbeispiel

Beispiel: Umrechnung von Farbräumen
Von RGB in YCbCr

Embedded Systems & Applications

A. Koch

Umrechnung RGB nach YCbCr

8b-Genauigkeit

Embedded Systems & Applications

Gleitkommadarstellung

A. Koch

$$
\begin{aligned}
y &= 0.299r + 0.587g + 0.114b \\
c_b &= -0.169r - 0.331g + 0.5b + 128 \\
c_r &= 0.5r - 0.419g - 0.082b + 128
\end{aligned}
$$

16b-Fixpunkt-Darstellung

$$
\begin{aligned}
y &= (77r + 150g + 29b)/256 \\
c_b &= (-43r - 85g + 128b + 32768)/256 \\
c_r &= (128r - 107g - 21b + 32768)/256
\end{aligned}
$$

C-Code für Farbraumumrechnung

Embedded Systems & Applications

A. Koch

Color Conversion Function:

```
void rgb2ycc(
  signed char r, signed char g, signed char b,
  signed char *y, signed char *cb, signed char *cr)
{
   *y  = (  77*r + 150*g  +   29*b           ) >> 8;
  *cb = ( -43*r  -   85*g  + 128*b + 32768) >> 8;
  *cr  = (128*r -  107*g   -   21*b + 32768) >> 8;
}
```

Program Loop:

```
for (…) {
     /* Convert 1 RGB Pixel to 1 YCbCr pixel */
   rgb2ycc(RGB[i], RGB[i+1], RGB[i+2], &YCC[i], &YCC[i+1], &YCC[i+2]);
}}
```

Laufzeit der reinen Software-Lösung

Embedded Systems & Applications

A. Koch

Software	WR	ISEF (Bit-Width)	ISEF (State Reg.)	Instruction Pipeline	Cycle (K Cycles)	Factor
RGB2YCC ANSI – C Only				√	3458	1

Farbraumumrechnung in Stretch-C

Definition einer neuen Instruktion auf ISEF durch Fusion

Embedded Systems & Applications

A. Koch

Color Conversion Function:

```
SE_FUNC /* Tells Stretch C-Compiler to reduce this function to an instruction */
void rgb2ycc(WR A, WR *B)  /* Data Bandwidth – Move 24 bits in Single Register */
{
  se_sint<8> r, g, b, y, cb, cr;
  int i;
     r = A(7,0); g = A(15,8); b = A(24,16); /* Extract Data; No Compute Cycles */

     y  = (  77*r + 150*g  +  29*b          ) >> 8;
    cb = ( -43*r  -  85*g  +  128*b + 32768) >> 8;
    cr  = ( 128*r - 107*g  -  21*b +  32768) >> 8;

   *B = (cr,cb, y);            /* pack YCbCr to B; No Compute Cycles*/
}
```

Program Loop:

```
for (…) {
     WRGET0(&A, 3);       /* Load 3 bytes (1 RGB pixels)  */
     RGB2YCC(A, &B);    /* Convert 1 pixel              */
     WRPUT0(B, 3);        /* Store 3 bytes (1 YCbCr pixel */
}
```

- C-artige Schreibweise statt Verilog-artigem TIE
- Basiskontrukte ähnlich

Laufzeit mit fusionierter Instruktion auf ISEF

Embedded Systems & Applications

A. Koch

Software	WR	ISEF (Bit-Width)	ISEF (State Reg.)	Instruction Pipeline	Cycle (K Cycles)	Acc. Factor
RGB2YCC ANSI – C Only				√	3458	1
Operator Fusion		√	√	√	219	15

Verfeinerung der ISEF-Instruktion durch SIMD

Embedded Systems & Applications

A. Koch

Color Conversion Function:

```
SE_FUNC  /* Extension instruction converting pixels */
void rgb2ycc(WR A, WR *B)  {      /* Data Bandwidth – Move 120 bits */
  se_sint<8> r[5], g[5], b[5], y[5], cb[5], cr[5];
  int i, j;
  /* Unpack A to RGB Data, Does Not Use Any Compute Cycles */
  for (i = 0; i < 5; i++, j = i*24) { r[i] = A(j+7, j); g[i] = A(j+15, j+8); b[i] = A(j+23, j+16) }
  /* Convert 5 pixels */
  for (i = 0; i < 5; i++) {
     y[i]  = (  77*r[i] + 150*g[i] +   29*b[i]          ) >> 8;
    cb[i] = (-43*r[i] -    85*g[i] + 128*b[i] + 32768) >> 8;
    cr[i]  = (128*r[i] -  107*g[i] -   21*b[i] + 32768) >> 8;
  }  /* pack YCbCr to B; Does Not Use Any Compute Cycles */
  *B = (cr[4],cb[4],y[4],cr[3],cb[3],y[3],cr[2],cb[2],y[2],cr[1],cb[1],y[1], cr[0],cb[0],y[0]);
}
```

Program Loop:

```
for (…) {
    WRGET0(&A, 15);       /* Load 15 bytes (5 RGB pixels)  */
    RGB2YCC(A, &B);     /* Convert 5 pixels              */
    WRPUT0(B, 15);         /* Store 15 bytes (5 YCbCr pixels */
}
```

- Rechnet auf 5 RGB Pixeln gleichzeitig

Laufzeit mit fusionierter SIMD-Instruktion auf ISEF

Embedded Systems & Applications

A. Koch

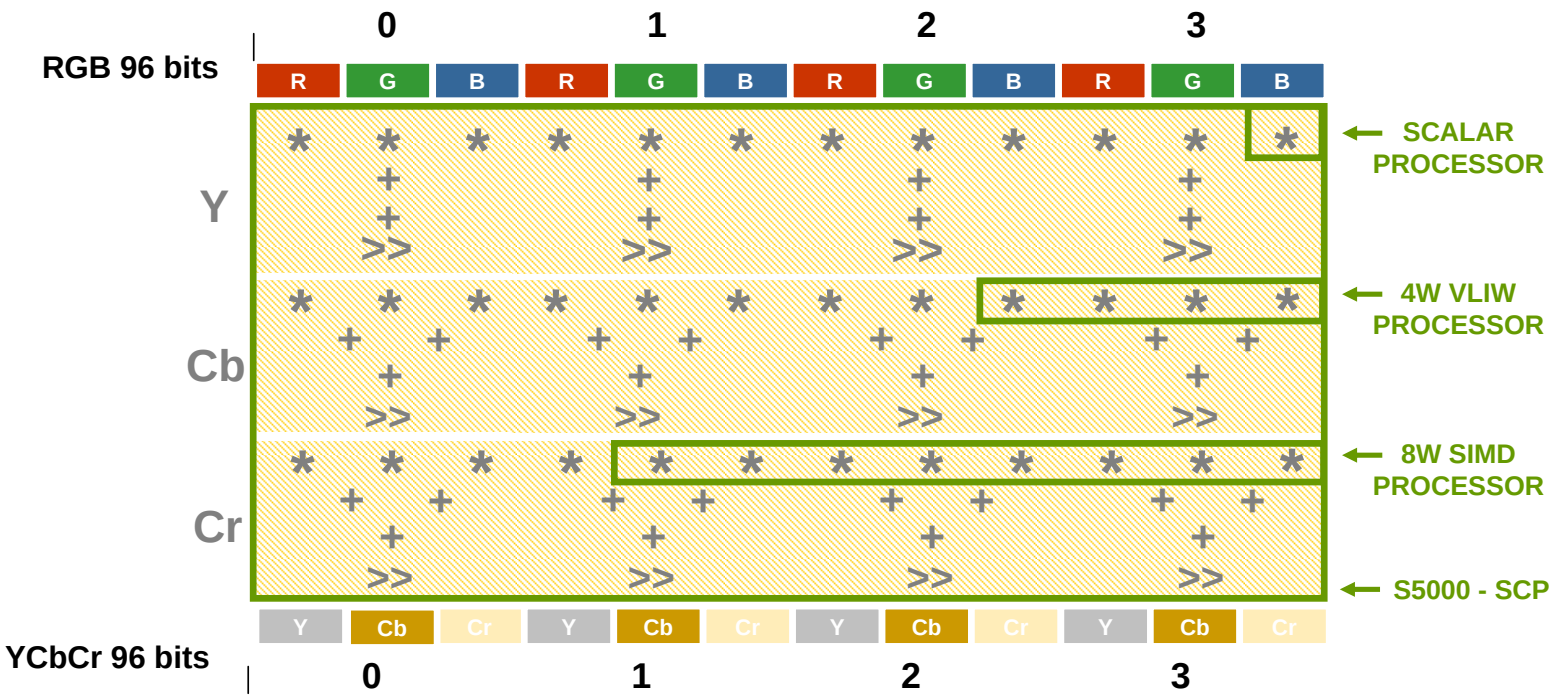
Software	WR	ISEF (Bit-Width)	ISEF (State Reg.)	Instruction Pipeline	Cycle (K Cycles)	Factor
RGB2YCC ANSI – C Only				√	3458	1
Operation Fusion		√	√	√	219	15
Vectorization	√	√	√	√	42	80

Nun 80x schneller als ursprüngliche C Version!

Grad der Parallelität bei unterschiedlichen Technologien

Annahme hier: 96b breite Datenworte

A. Koch



Nichtausgerichtete Speicherzugriffe

- Zugriff auf bis zu 128b breite Daten als Byte-Folgen
- Daten können beliebig im Speicher ausgerichtet sein
  - Unterschied zu direkten WRF Load/Store-Anweisungen
- Speicherströme (*streams*) für reguläre Zugriffe
  - Drei lesende, einen schreibenden
- **`INIT`**-Funktion programmiert je Stream
  - Startaddresse
  - Richtung (aufsteigende oder absteigende Adressen)
- **`GET`**/**`PUT`** liest/schreibt Daten in/aus WR-Register

Beispiel: Lesen von Byte-Daten

Über Lesestrom 0

A. Koch

```
#define INCR 0
unsigned char inArray[1024];
WR X;

WRGET0INIT(INCR, inArray);    // init0 (mode, addr), init1
…
WRGET0I(&X, 6);               // load next 6 bytes into X
…
WRGET0I(&X, 9);               // load next 9 bytes into X
```

- Annahme:
  `inArray[] = { 0x01,0x02,0x03,0x04,0x05, ...}`
- Nach dem erstem **`WRGET0I`** hat WR-Register **`X`** den Wert 0x060504030201
- Nach dem zweiten **`WRGET0I`**: 0x0F0E0D0C0B0A090807

Beispiel: Schreiben von Byte-Daten

Über den einzigen Schreibstrom

A. Koch

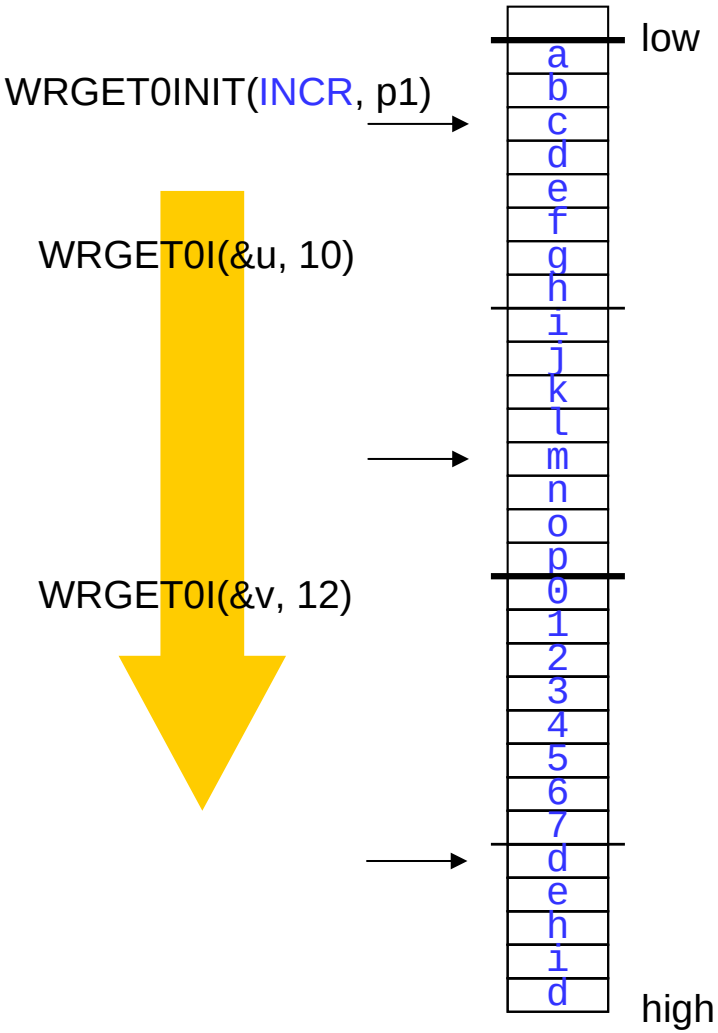
```
#define INCR 0
unsigned char outArray[1024];
WR X;

WRPUTINIT(INCR, outArray);    // init 0
…
WRPUTI(X, 6);                 // store 6 bytes from X
…
WRPUTI(X, 9);                 // store 9 bytes from X
…
WRPUTFLUSH();                 // flush0, flush1
```

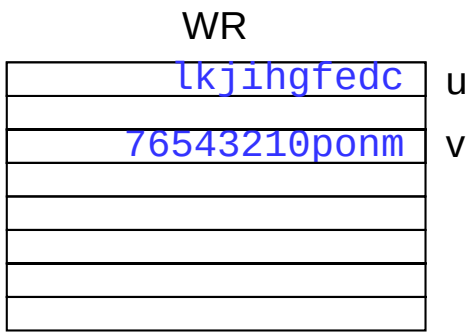
- Annahme beim ersten Aufruf: **`x`**=0x665544332211
- Annahme beim zweiten Aufruf **`x`** =0xFFEEDDCCBBAA998877
- **`outArray`** beginnt danach mit 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88, . . .

Aufsteigende Adressen mit Post-Inkrement

Register wird von rechts nach links gefüllt

A. Koch



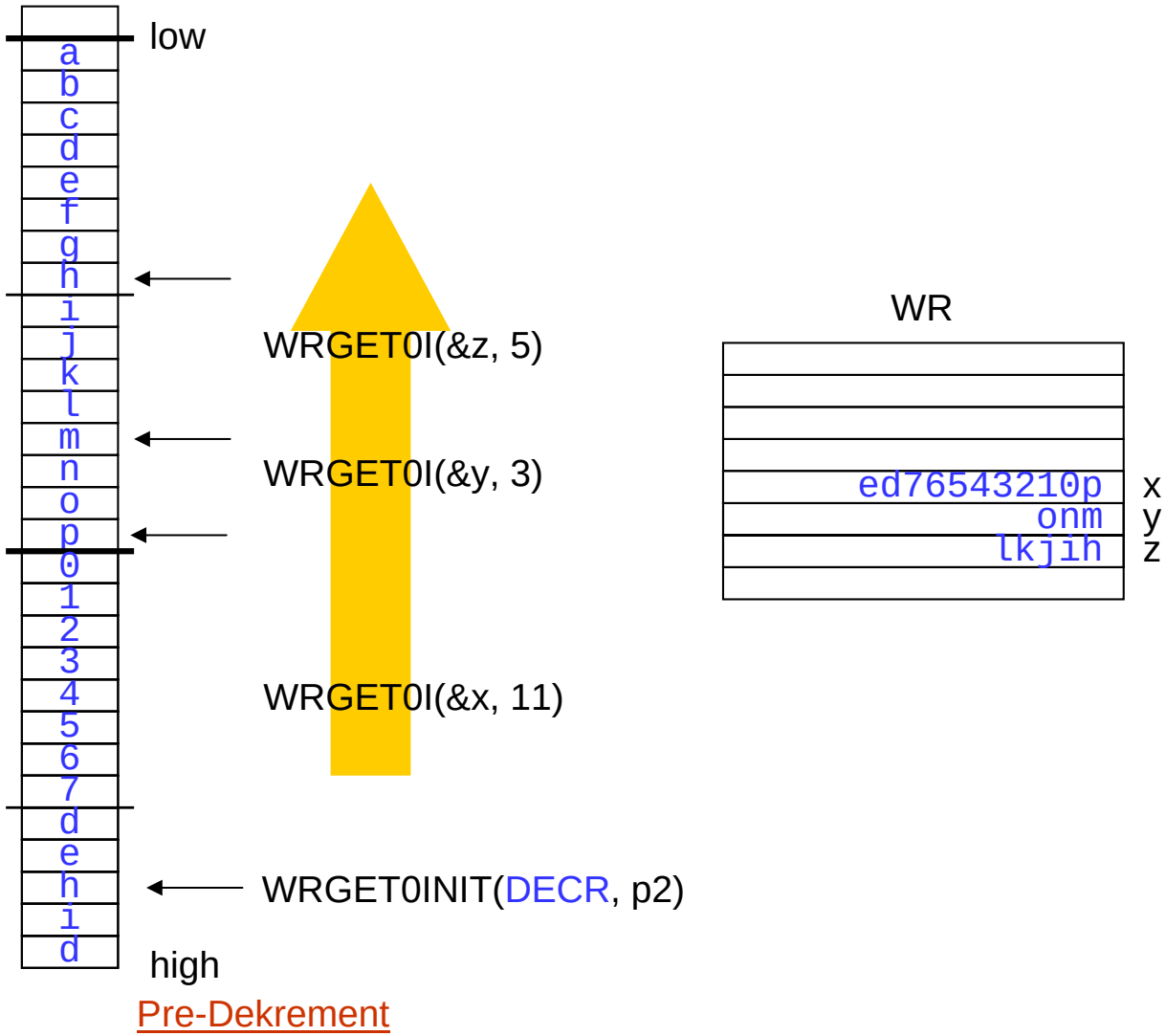


Post-Inkrement

Absteigende Adressen mit Pre-Dekrement

Register wird von links nach rechts gefüllt

A. Koch



Pre-Dekrement

Übersicht über Streaming-Befehle

Embedded Systems & Applications

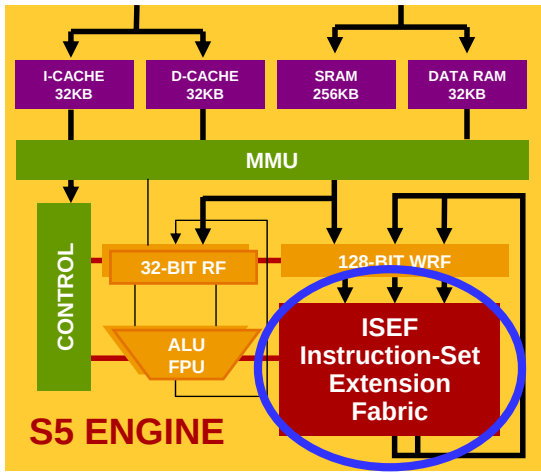
A. Koch

Bank	Operation	Stream No. (GET only)	Mode
WR WRA WRB	GET (load) PUT (store)	0 1 2	I (immediate byte count and advance amount) X (register byte count and advance amount) XX (register specifies separate byte count and advance amounts)

- Daten stehen immer in der Reihenfolge ihres Auftretens im Speicher im Register.
- Angaben über Länge der Leseoperation auch aus Register möglich
- Auch unterschiedliche Schrittweiten möglich
  - 4 Bytes lesen, nächste Leseadresse aber um 8 Bytes erhöhen
- Ähnliche Befehle lesen/schreiben auf Bit-Ebene
  - Bis zu 32b in einer Operation lesbar

Instruction-Set Extension Fabric

Kann als konfigurierbare ALU angesehen werden



A. Koch

- Maximal 48 Bytes Operanden (drei 128b WR-Register)
- Maximal 32 Bytes Ergebnis (zwei 128b WR-Register)
- Pipelined, multi-zyklen Operationen realisierbar
- S5 Engine hat zwei ISEF RAs
  - Teilen sich dieselben Register
  - Nur ein ISEF gleichzeitig aktiv
  - Vorteile: Mehr Platz und schnellere Rekonfiguration

Zwischenstand

- Rekonfigurierbare Prozessoren
- Stretch S5000-Architektur
- Erstes Beispiel in Stretch-C
- Schwerpunkt: Kommunikation mit ISEF
  - Wide Registers ausgerichtet lesen/schreiben
  - Streaming von unausgerichteten Daten