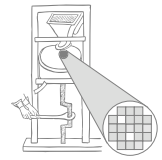


Eingebettete Prozessorarchitekturen

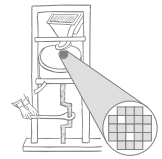
9. Entwurf mit rekonfigurierbaren Prozessoren

Andreas Koch

FG Eingebettete Systeme und ihre Anwendungen
Informatik, TU Darmstadt

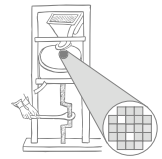
Wintersemester 2006/2007

Material und Literatur



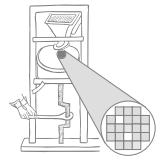
- Auszüge aus Trainingsmaterial der Fa. Stretch
  - Insbesondere alle Zeichnungen und technischen Daten
- Material ist vertraulich
  - Nur für die Lehre zur Verfügung gestellt
  - Darf nicht weiterverbreitet werden

Zwischenstand



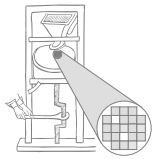
A. Koch

- Stretch S5000 Architektur
- Programmierung in Stretch-C
- Flächen- und Zeitabschätzungen



Pipelining

Unterschiedliche Taktbereiche
clock domains



- Feste Xtensa V Pipeline läuft mit max. 300 MHz
- Rekonfigurierbare ISEFs laufen mit max. 100 MHz
- Starte eine ISEF EI je zwei Xtensa V Instruktionen
- Xtensa Speichertransfers während EI-Ausführungszeit

Beispiel

Schleife entrollt

```
WRGET0I(&A, 15)
rgb2ycc(A, &B)
WRPUTI(B,15)
WRGET0I(&A, 15)
rgb2ycc(A, &B)
WRPUTI(B,15)
```

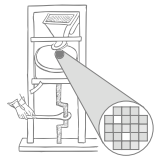
Spannender Teil

```
rgb2ycc(A, &B)
WRPUTI(B,15)
WRGET0I(&A, 15)
```

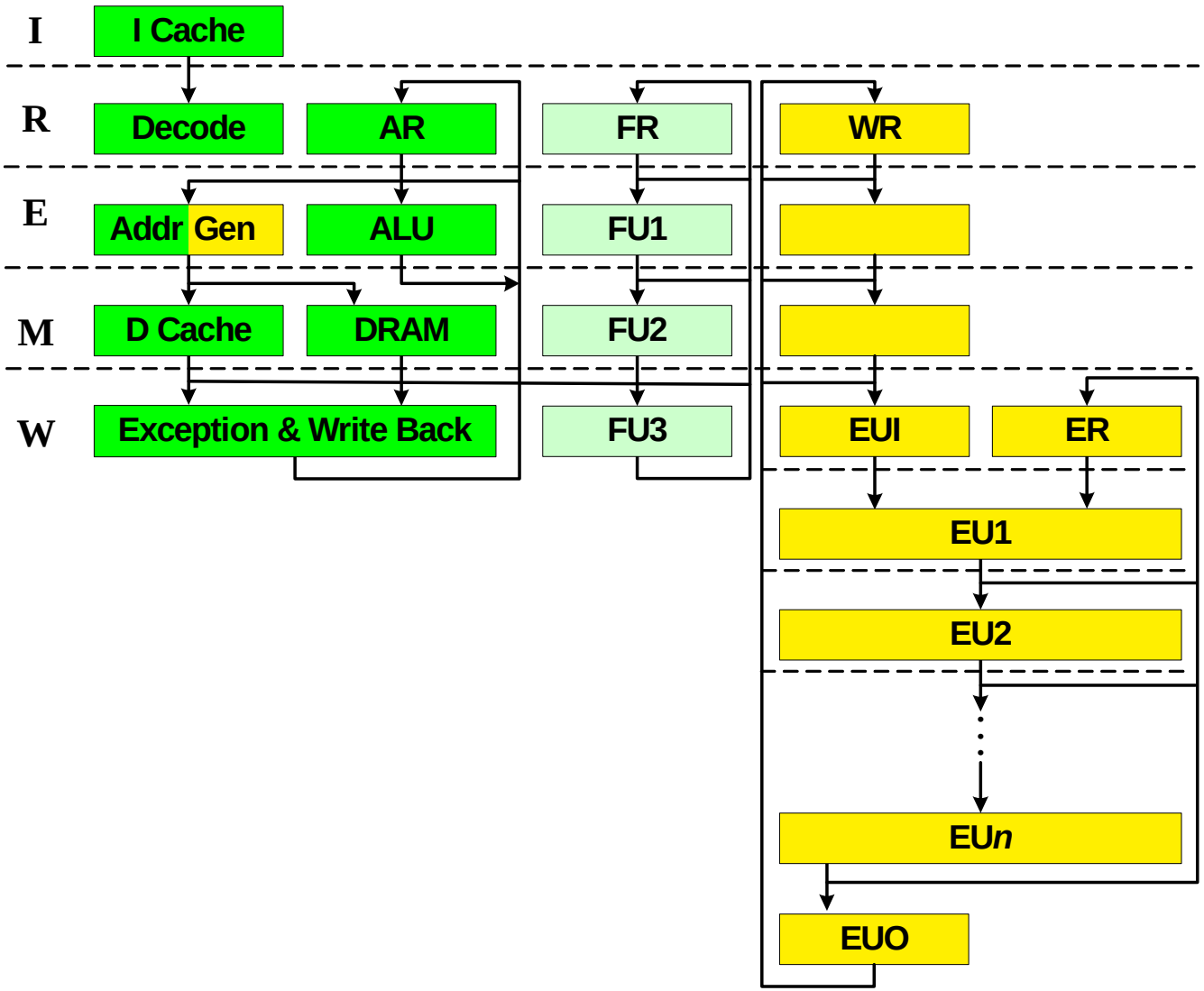
Während Ausführungszeit von **`rgb2ycc`** werden alte Daten geschrieben und neue geholt.

Pipeline-Struktur

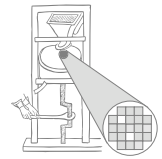
Bis zu 31 Takte lang



A. Koch

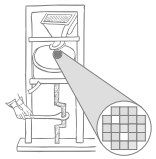


Pipeline-Ablauf im ISEF



- Operanden werden am Anfang von EUI-Stufe gelesen (*input*)
- Ergebnisse werden am Ende von EUO-Stufe ausgegeben (*output*)
- EI mit n-Takten hat Stufen EUI, EU1, . . . , EUn, EUO
- n ist hier in Xtensa-Takten, i.d.R. ISEF-Takte = $n/3$
- Extension Registers dürfen . . .
  - am Anfang jeder EUi-Stufe gelesen werden
  - am Ende jeder EUj-Stufe mi $j \geq i$ geschrieben werden
- Alle Register (AR, WR, ER) sind
  - Interlocked: ISEF-Pipeline hält an, wenn RAW verletzt
  - Bypassed: Register sofort nach Definition verwendbar

Beispiel Pipelining



A. Koch

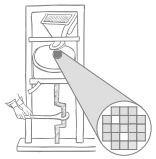
```
for (i = 0; i < NP/5; i++) {
    WRGET0I(&A, 15); // load available after M
    rgb2ycc(A, &B);  // rgb2ycc's latency is 10 cycles (2*3 + 4)
    WRPUTI(B, 15);   // B must be available before E
}
```

Compute-store stall for B

Note Zero Overhead Loop
Each iteration takes 12 cycles

n		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	WRGET0I	I	R	E	M	W																
0	rgb2ycc		I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO								
0	WRPUTI			I	R	-	-	-	-	-	-	-	-	-	E	M	W					
1	WRGET0I				I	-	-	-	-	-	-	-	-	-	R	E	M	W				
1	rgb2ycc														I	R	E	M	EUI	EU1	EU2	EU3

Komplizierteres Szenario mit Loop Unrolling



A. Koch

Unrolling im Compiler abgeschaltet:

Stalls highlighted

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
movi a8, 128	I	R	E	M	W																
wrputinit 0, a3		I	R	E	M	W															
wrget0init0 0, a2			I	R	E	M	W														
wrget0init1				I	R	E	M	W													
loopgtz a8, d00015t					I	R	E	M	W												
wraget0i wra0, 15						I	R	E	M	W											
se_rgb2ycc wra0, w							I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO			
wraputi wra0, 15								I	R	R	R	R	R	R	R	R	R	R	E	M	W

Profiler: Static

Pipeline Analyzer

3 instructions
= 12 cycles
(9 stalls)

Zero-overhead loop instruction

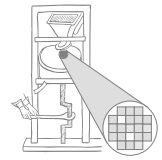
Unrolling erlaubt:

12 instructions
= 12 cycles
(0 stalls)

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
se_rgb2ycc wra3, wra3	I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO												
loopgtz a8, d00015bc		I	R	E	M	W																		
wraget0i wra0, 15			I	R	E	M	W																	
se_rgb2ycc wra4, wra5				I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO									
wraget0i wra2, 15					I	R	E	M	W															
wraputi wra7, 15						I	R	E	M	W														
se_rgb2ycc wra6, wra7							I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO						
wraputi wra1, 15								I	R	E	M	W												
wraget0i wra4, 15									I	R	E	M	W											
se_rgb2ycc wra0, wra1										I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO			
wraputi wra3, 15											I	R	E	M	W									
wraget0i wra6, 15												I	R	E	M	W								
se_rgb2ycc wra2, wra3													I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO
wraputi wra5, 15														I	R	E	M	W						

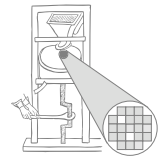
Profiler: Static

Pipeline Analyzer



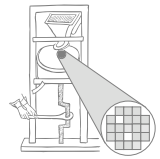
Loop Unrolling

Idee beim Loop Unrolling



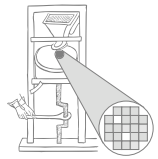
- Beobachtung
  - Latenz einer n-Takte EI ist $n + 2$ Takte (EUI+EUO)
  - n-Takte EI gefolgt von STORE hat Latenz $n + 4$
- Idee
  - Ur-Schleife von N Iterationen entrollen um Faktor M
  - Nun M Datenelemente je Iteration rechnen
  - Dafür aber nur noch N/M-Iterationen ausführen
  - Wartezyklen mit sinnvollen Operationen füllen
    - Jetzt genug im Schleifenrumpf vorhanden

Rezept für Loop Unrolling



1. M Kopien des Rumpfes in Schleife anlegen
   - Jede mit eigenem Satz Variablen
   - Verschiebe STORE-Anweisungen genau vor Berechnung
   - Nun Grenzen auf $N/M - 1$ Iterationen setzen
     - Eine Iteration realisiert in Prolog/Epilog
2. Kopiere neuen Rumpf vor Schleife als Prolog
   - Lösche alle STOREs in Prolog
3. Kopiere neuen Rumpf hinter Schleife als Epilog
   - Lösche dabei alles ausser STOREs

Beispiel: `rgb2ycc`-Schleife



A. Koch

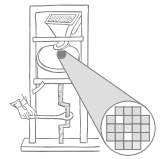
```
for (i = 0; i < NP/5; i++) {
    WRGET0I(&A, 15); // load available after M
    rgb2ycc(A, &B);  // rgb2ycc's latency is 10 cycles (2*3 + 4)
    WRPUTI(B, 15);   // B must be available before E
}
```

Compute-store stall for B

Note Zero Overhead Loop
Each iteration takes 12 cycles

n		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
0	WRGET0I	I	R	E	M	W																
0	rgb2ycc		I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO								
0	WRPUTI			I	R	-	-	-	-	-	-	-	-	-	E	M	W					
1	WRGET0I				I	-	-	-	-	-	-	-	-	-	R	E	M	W				
1	rgb2ycc														I	R	E	M	EUI	EU1	EU2	EU3

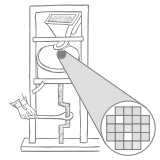
Beispiel: Schleife entrollt um $M = 4$



A. Koch

```
WRGET0I(&A0, 15);           // load A0
rgb2ycc(A0, &B0);           // convert A0
WRGET0I(&A1, 15);           // load A1
rgb2ycc(A1, &B1);           // convert A1
WRGET0I(&A2, 15);           // load A2
rgb2ycc(A2, &B2);           // convert A2
WRGET0I(&A3, 15);           // load A3
rgb2ycc(A3, &B3);           // convert A3

for (i = 1; i < NPIX5/4; i++) {
    WRGET0I(&A0, 15);       // load A0
    WRPUTI(B0, 15);         // store B0
    rgb2ycc(A0, &B0);       // convert A0->B0

    WRGET0I(&A1, 15);       // load A1
    WRPUTI(B1, 15);         // store B1
    rgb2ycc(A1, &B1);       // convert A1->B1

    WRGET0I(&A2, 15);       // load A2
    WRPUTI(B2, 15);         // store B2
    rgb2ycc(A2, &B2);       // convert A2->B2

    WRGET0I(&A3, 15);       // load A3
    WRPUTI(B3, 15);         // store B3
    rgb2ycc(A3, &B3);       // convert A3->B3
}

WRPUTI(B0, 15);             // store B0
WRPUTI(B1, 15);             // store B1
WRPUTI(B2, 15);             // store B2
WRPUTI(B3, 15);             // store B3
```

Beispiel: Details des Entrollens



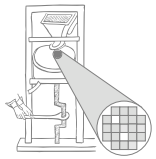
i starts at 1, goes to max/4

A. Koch

```
WRGET0I(&A0, 15);           // load A0
rgb2ycc(A0, &B0);           // convert A0
WRGET0I(&A1, 15);           // load A1
rgb2ycc(A1, &B1);           // convert A1
WRGET0I(&A2, 15);           // load A2
rgb2ycc(A2, &B2);           // convert A2
WRGET0I(&A3, 15);           // load A3
rgb2ycc(A3, &B3);           // convert A3

for (i = 1; i < NPIX5/4; i++) {
    WRGET0I(&A0, 15);       // load A0
    WRPUTI(B0, 15);         // store B0
    rgb2ycc(A0, &B0);       // convert A0->B0

    WRGET0I(&A1, 15);       // load A1
    WRPUTI(B1, 15);         // store B1
    rgb2ycc(A1, &B1);       // convert A1->B1

    WRGET0I(&A2, 15);       // load A2
    WRPUTI(B2, 15);         // store B2
    rgb2ycc(A2, &B2);       // convert A2->B2

    WRGET0I(&A3, 15);       // load A3
    WRPUTI(B3, 15);         // store B3
    rgb2ycc(A3, &B3);       // convert A3->B3
}

WRPUTI(B0, 15);             // store B0
WRPUTI(B1, 15);             // store B1
WRPUTI(B2, 15);             // store B2
WRPUTI(B3, 15);             // store B3
```

```
WRGET0I(&A0, 15);           // load A0
```

Beispiel: Ergebnis



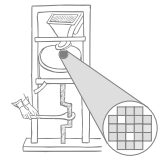
A. Koch

Profiler: Static

Instruction	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
loopgtz a8, d000157d	I	R	E	M	W																			
wraputi wra3, 15		I	R	E	M	W																		
wraget0i wra3, 15			I	R	E	M	W																	
se_rgb2ycc wra3, wra3				I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO									
wraputi wra2, 15					I	R	E	M	W															
wraget0i wra2, 15						I	R	E	M	W														
se_rgb2ycc wra2, wra2							I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO						
wraputi wra1, 15								I	R	E	M	W												
wraget0i wra1, 15									I	R	E	M	W											
se_rgb2ycc wra1, wra1										I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO			
wraputi wra0, 15											I	R	E	M	W									
wraget0i wra0, 15												I	R	E	M	W								
se_rgb2ycc wra0, wra0													I	R	E	M	EUI	EU1	EU2	EU3	EU4	EU5	EU6	EUO

Pipeline Analyzer

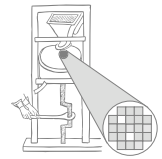
- Nun keine Stall-Zyklen mehr
- Vorher: 12 Takte je Iteration
- Nun: 12 Takte je 4 Iterationen



Optimierung von EIs

Beispiel: Maximaler 16b-Wert in Array

EI bearbeitet je 8 Werte

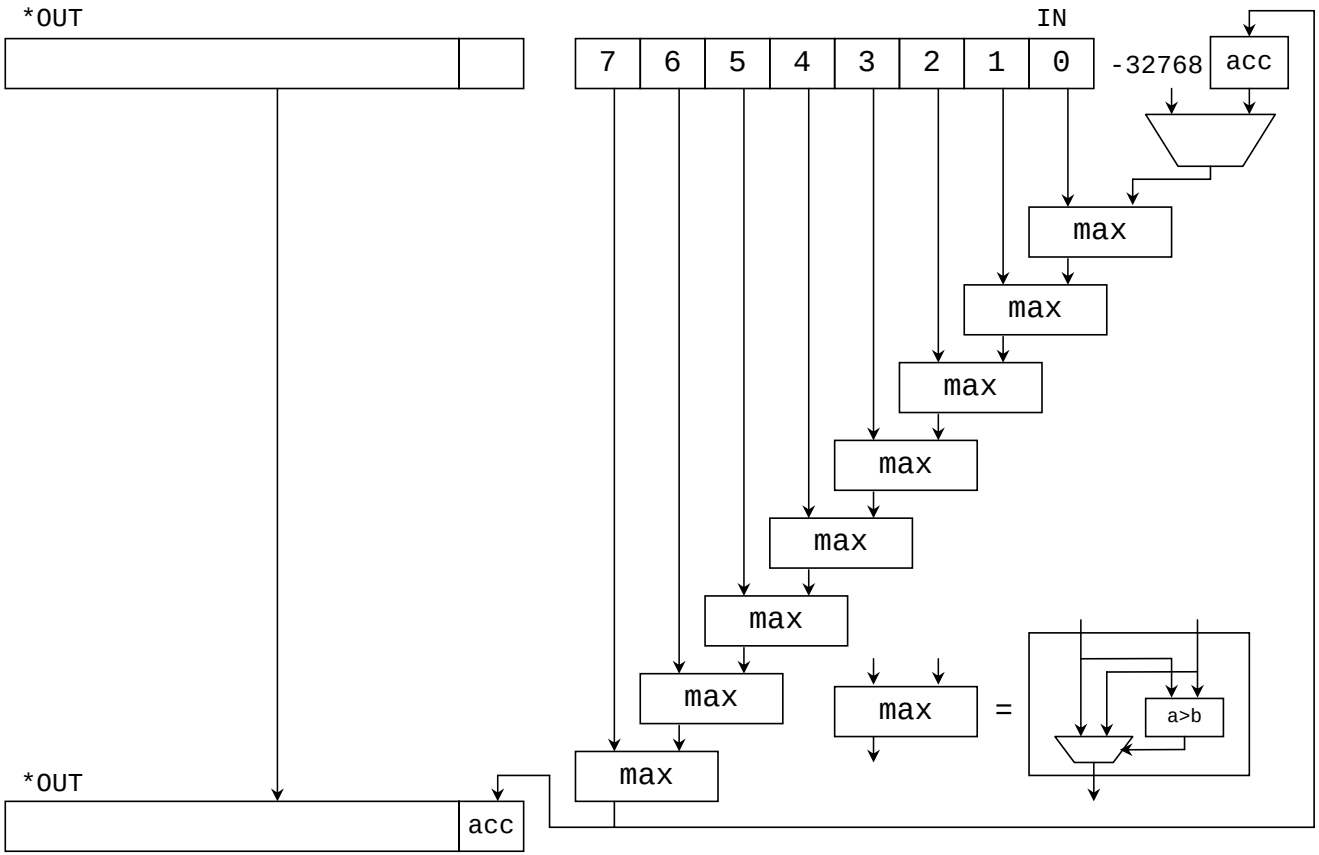


A. Koch

```
#define N (8)
SE_FUNC void runningmax ( SE_INST MAX8_INIT, SE_INST MAX8, WRA IN, WRA *OUT )
{
    int i;
    static se_sint<16> acc;
    se_sint<16> x[N];

    for (i = 0; i < N; i++) x[i] = IN(i*16 + 15, i*16);

    acc = MAX8_INIT ? -32768 : acc;
    for (i = 0; i < N; i++) {
        if (x[i] > acc)   acc = x[i];
    }

    (*OUT)(15,0) = acc;
}
```

```
INSTRUCTION=runningmax:MAX8 Opcode: 1: Assembly 'se_max8 IN:in, OUT:in+OUT:out'
 Input : variable 'IN' read from WRA on ISEF read port 1 at beginning of cycle 1
 Input : variable 'OUT' read from WRA on ISEF read port 0 at beginning of cycle 1
 Output: variable 'OUT' written to WRA on ISEF write port 0 at end of cycle 5
 ISEF Local State variable 'acc.0' (16 bits) read at beginning of cycle 2
 ISEF Local State variable 'acc.0' (16 bits) written at end of cycle 4
                                                        (Write-Read = 2)
```

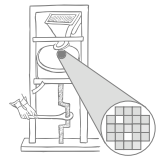
Beispiel: Erzeugte Hardware



A. Koch

- `*OUT` wird versehentlich auch gelesen
- Berechnung von **`max`** erfolgt hier seriell
- Langer Zeitraum zwischen Lesen/Schreiben von **`acc`**
  - Potentiell lange Interlock-Delays → niedriger Durchsatz

Beispiel: Verbesserte Version

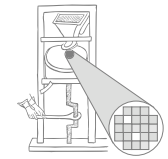


A. Koch

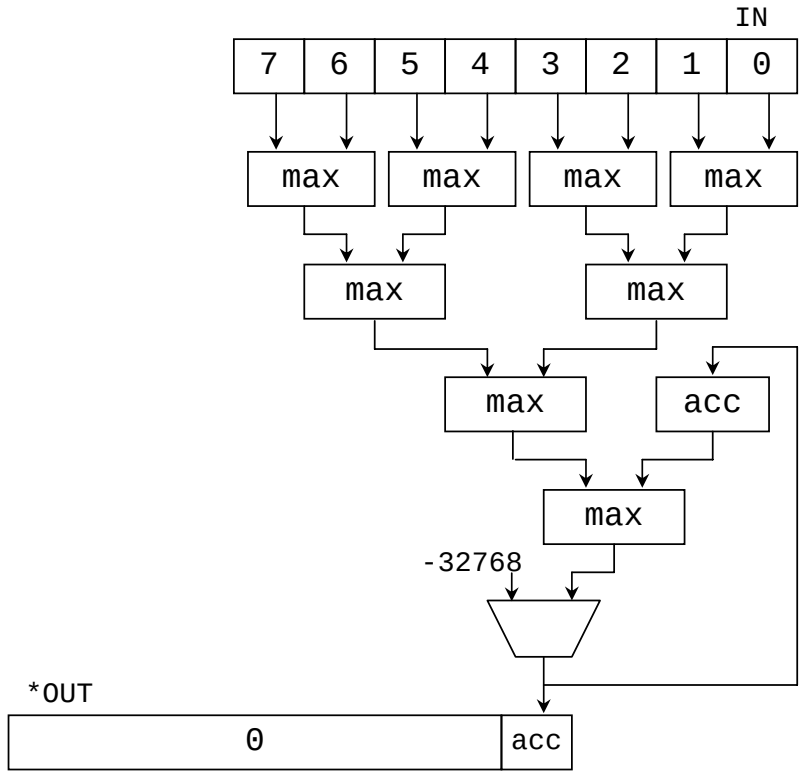
```
#define MYMAX(a, b) (((a) > (b)) ? (a) : (b))
#define N (8)
SE_FUNC void somecode ( SE_INST MAX8_INIT, SE_INST MAX8, WRA IN, WRA *OUT )
{
    int i;
    static se_sint<16> acc;
    se_sint<16> max, x[N];

    for (i = 0; i < N; i++) x[i] = IN(i*16 + 15, i*16);

    max = MYMAX( MYMAX( MYMAX(x[0], x[1]), MYMAX(x[2], x[3]) ),
                 MYMAX( MYMAX(x[4], x[5]), MYMAX(x[6], x[7]) ) );
    acc = MAX8_INIT ? -32768 : MYMAX(max, acc);

    (*OUT) = 0;
    (*OUT)(15,0) = acc;
}
```

```
INSTRUCTION=somecode:MAX8 Opcode: 1: Assembly 'se_max8 IN:in, OUT:out'
 Input : variable 'IN' read from WRA on ISEF read port 1 at beginning of cycle 1
 Output: variable 'OUT' written to WRA on ISEF write port 0 at end of cycle 5
 ISEF Local State variable 'acc.0' (16 bits) read at beginning of cycle 3
 ISEF Local State variable 'acc.0' (16 bits) written at end of cycle 3
                                                    (Write - Read = 0)
```

Beispiel: Verbesserte Hardware

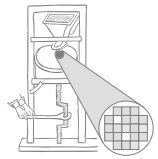


A. Koch



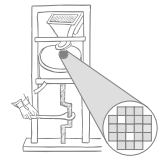
- Oberste 112b von **`*OUT`** nun nicht mehr gelesen
  - Direkt auf 0 gesetzt
- **`max`**-Berechnung erfolgt weitgehend parallel
- Nun minimales Zeitintervall Lesen/Schreiben **`acc`**
  - Keine Interlock-Delays mehr → höherer Durchsatz

Allgemeine Tipps



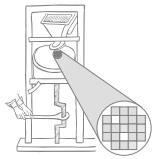
- Setze Variablen auf passende Größe
  - Beispiel: `se_sint<18>` statt `int`
- Spezifiziere alle Alternativen bei Verzweigungen
  - Zu jedem `if` ein `else`
  - In jedem `switch` ein `default`
  - ➥ähnlich zu Verilog (kommt noch!)
- Setze alle Bits in Variablen

Konstantentabellen in Stretch-C



- Werden in verschiedensten Situationen eingesetzt
  - Ersetze Berechnung durch Nachschlagen (*lookup*)
  - $\sin x$, $\log x$, ...
  - Routing-Tabellen
  - Zustandsautomat (in mehreren EI-Aufrufen ausgeführt)
- Im ISEF nicht durch Speicher (RAM/ROM) realisiert
- Sondern durch kombinatorische Logik
  - Logikminimierung möglich (ggf. sehr rechenaufwendig!)

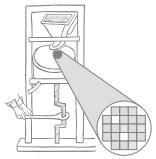
Konstantentabellen 1
Beispiel



A. Koch

```
// GF 8-bit inverse and affine for AES encryption
static const char gf_invaffine_enc[256] = {
     0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
     ...
     0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};

SE_FUNC void AES_TABLE_LOOKUP( WRA in, WRA *out )
{
     int  i;
     se_uint<8> x, y;

     *out = 0;
     for ( i = 3;  i >= 0;  i-- ) {
          x = in(8*i+7, 8*i);                    /* PIPELINE=108 LOGIC=696 MUX=633 CYCLE=5  */
          y = gf_invaffine_enc[integer(x)];      /* PIPELINE=36 LOGIC=106 MUX=207 CYCLE=5  */
          *out = (*out, y);
     }
}

/*                Final resource usage report                          */
/* Total AUs = 1759 out of 4096                                        */
/* Total MUs = 0 out of 8192                                           */
```

Beispiel: Konstantentabellen 1
Diskussion



A. Koch

```
// GF 8-bit inverse and affine for AES encryption
static const char gf_invaffine_enc[256] = {
     0x63, 0x7c, 0x77, 0x7b, 0xf2, 0x6b, 0x6f, 0xc5, 0x30, 0x01, 0x67, 0x2b, 0xfe, 0xd7, 0xab, 0x76,
     0xca, 0x82, 0xc9, 0x7d, 0xfa, 0x59, 0x47, 0xf0, 0xad, 0xd4, 0xa2, 0xaf, 0x9c, 0xa4, 0x72, 0xc0,
     0xb7, 0xfd, 0x93, 0x26, 0x36, 0x3f, 0xf7, 0xcc, 0x34, 0xa5, 0xe5, 0xf1, 0x71, 0xd8, 0x31, 0x15,
     ...
     0x8c, 0xa1, 0x89, 0x0d, 0xbf, 0xe6, 0x42, 0x68, 0x41, 0x99, 0x2d, 0x0f, 0xb0, 0x54, 0xbb, 0x16
};

SE_FUNC void AES_TABLE_LOOKUP( WRA in, WRA *out )
{
     int  i;
     se_uint<8> x, y;

     *out = 0;
     for ( i = 3;  i >= 0;  i-- ) {
          x = in(8*i+7, 8*i);
          y = gf_invaffine_enc[integer(x)];
          *out = (*out, y);
     }
}

/*                Final resource usage report                    */
/* Total AUs = 1759 out of 4096                                  */
/* Total MUs = 0 out of 8192                                     */
```

Just 1759 AUs to implement **four** instances of a lookup table:

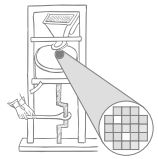
- each table has **256** entries, **8-bits** per entry
- entry values are ~**random**

If implemented in RAM, this would take 8192 bits

```
/* PIPELINE=108 LOGIC=696 MUX=633 CYCLE=5  */
/* PIPELINE=36 LOGIC=106 MUX=207 CYCLE=5  */
```

Beispiel: Konstantentabellen 2

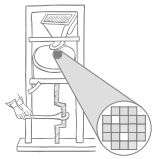
Beispiel



A. Koch

```
#define NBITS (12)
#define TABLESIZE (1 << NBITS)
#define INITIAL_REMAINDER (0xFFFFFFFF)
#define FINAL_XOR_VALUE (0xFFFFFFFF)

static se_uint<32> crcTable[TABLESIZE] = {
    0x00000000, 0x04c11db7, 0x09823b6e, 0x0d4326d9,
    0x130476dc, 0x17c56b6b, 0x1a864db2, 0x1e475005,
    . . .
};
static se_uint<32> remainder;

SE_FUNC void crc32_func( SE_INST CRC32_INIT, SE_INST CRC32, unsigned int in, WRA *out )
{
     se_uint<NBITS> inbits = ((se_uint<NBITS>)in)(0,NBITS-1);          // Reverse the bits!
     se_uint<NBITS> data   = (se_uint<NBITS>)( inbits ^ (remainder >> (32 - NBITS)) );
     remainder = CRC32_INIT ? INITIAL_REMAINDER : (crcTable[integer(data)] ^ (remainder << NBITS));

     *out = remainder(0,31) ^ FINAL_XOR_VALUE;
}

/*                Final resource usage report                              */
/* Total AUs = 264 out of 4096                                             */
/* Total MUs = 0 out of 8192                                               */
```

Beispiel: Konstantentabellen 2
Diskussion



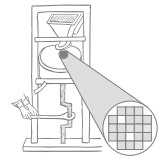
A. Koch

```
#define NBITS (12)
#define TABLESIZE (1 << NBITS)
#define INITIAL_REMAINDER (0xFFFFFFFF)
#define FINAL_XOR_VALUE (0xFFFFFFFF)

static se_uint<32> crcTable[TABLESIZE] = {
    0x00000000, 0x04c11db7, 0x09823b6e, 0x0d4326d9,
    0x130476dc, 0x17c56b6b, 0x1a864db2, 0x1e475005,
    . . .
};
static se_uint<32> remainder;

SE_FUNC void crc32_func( SE_INST CRC32_INIT, SE_INST CRC32, unsigned int in, WRA *out )
{
     se_uint<NBITS> inbits = ((se_uint<NBITS>)in)(0,NBITS-1);          // Reverse the bits!
     se_uint<NBITS> data   = (se_uint<NBITS>)( inbits ^ (remainder >> (32 - NBITS)) );
     remainder = CRC32_INIT ? INITIAL_REMAINDER : (crcTable[integer(data)] ^ (remainder << NBITS));

     *out = remainder(0,31) ^ FINAL_XOR_VALUE;
}

/*                Final resource usage report                          */
/* Total AUs = 264 out of 4096                                         */
/* Total MUs = 0 out of 8192                                           */
```

Just 264 AUs to implement lookup table:

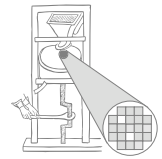
- table has **4096** entries, **32-bits** per entry
- entry values are ~**random**

If implemented in RAM, this would take 131,072 bits (500x larger)



Rekonfiguration

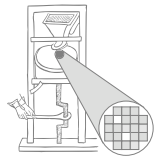
Rekonfiguration



- Name einer Konfiguration: Basisname der `.xc`-Datei
  - Also enthält `foo.xc` eine Konfiguration namens `foo`
- Rekonfigurationszeit für ISEF: ca. 100μs
- Automatisches und manuelles Rekonfigurieren

Automatische Rekonfiguration

On-Demand bei Ausführen einer EI der Konfiguration

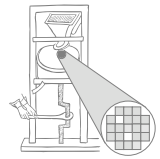


A. Koch

```
#include <s5000/sx-isef.h>
#include “isef_instruction.h"

void main() {
  WRA v1, v2, vOut;

    /* Auto Loading Enabled By Default */

    /* Use Instruction */
    FIR(v1, v2, &vOut);

    /*
     * If FIR doesn’t exist in either of the ISEF, processor will take an exception.
     * The Exception Handler will
     *    locate the ISEF configuration with the given instruction
     *    decide which ISEF to load based on the LRU Bit set in hardware
     *    start the ISEF loading
     * Upon completion of the load; it returns from the exception
     *
     * If the ISEF was previously configured, the handler will first invalidate
     * the old configuration.
     */

    /* At this point the FIR Instruction is executed by the processor */

    }
```

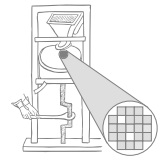
Manuelle Rekonfiguration

Explizites Laden der Konfiguration, auch vor Benutzung einer EI



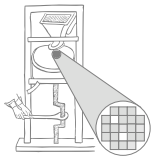
A. Koch

```
void main() {
  WRA v1, v2, vOut;

          /* Usage of Non Blocking API */
          error = sx_isef_load_by_name_async(sx_isef_a, “fir");

          /* Do Something – Typically takes 100 usecs to load the ISEF */
          process();

          /* Use FIR Instruction */
          FIR(v1, v2, &vOut);

          /*
           * For the first use of the FIR instruction the processor will take an exception
           * The Exception Handler will –
           *
           *  If “fir” configuration is still loading; the exception handler waits for the
           *  load to complete and then sets up the ISEF interface registers and exits out
           *  of the exception handler
           *
           *  If “fir” configuration is already loaded, the exception handler sets up the
           *  ISEF interface registers and exits out of the exception handler
           */
}
```

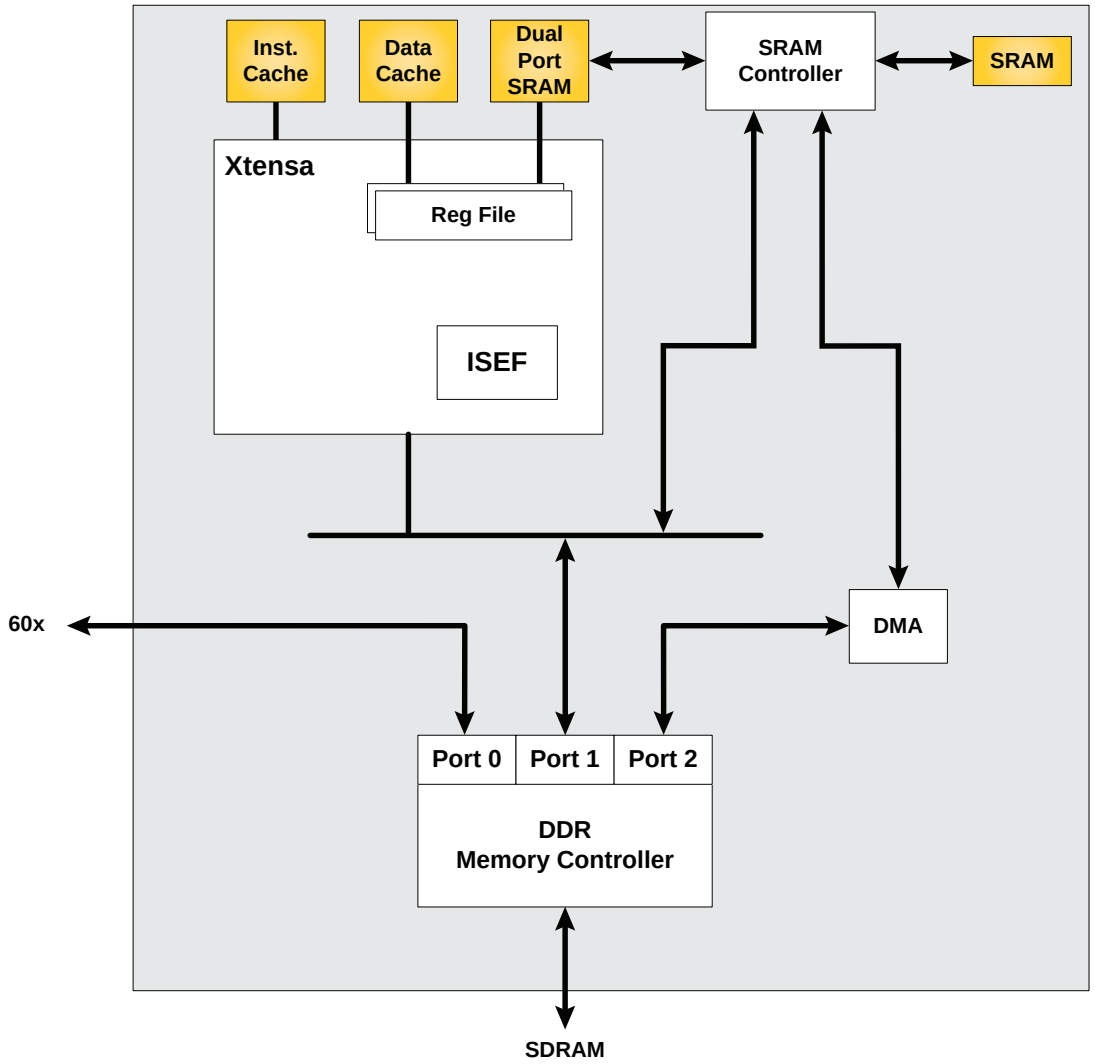


Speicher und DMA

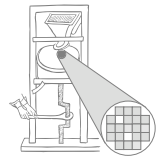
Übersicht über Speicher-Ressourcen



A. Koch



Übersicht über Speicher-Ressourcen



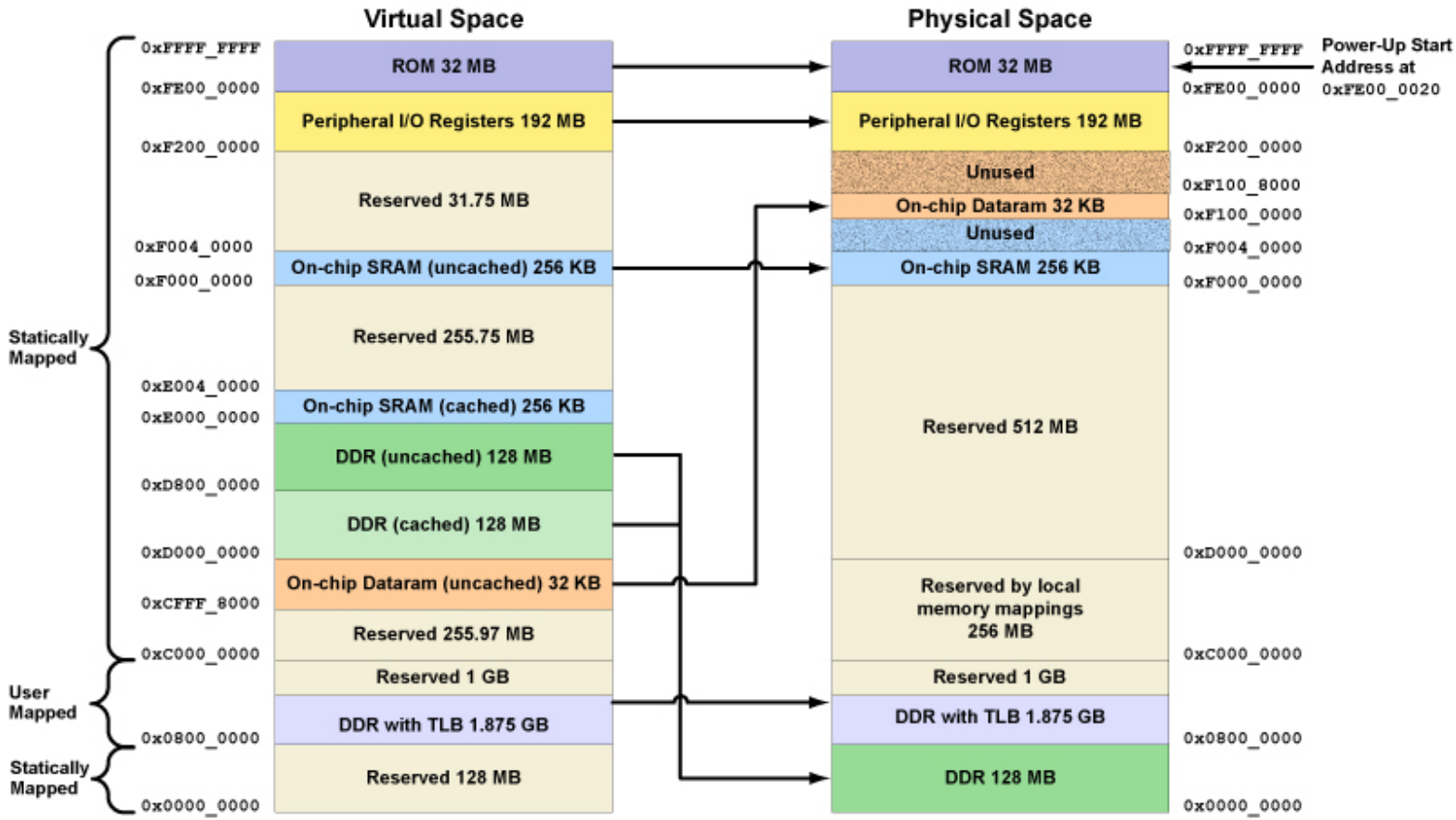
- Daten-Cache: 32KB
  - “Write-Back”-Betriebsart
  - Auch Schreib-Zugriffe gecached
- Dual-Port-Data-RAM (DPDR): 32KB
  - Liefert Daten in einem Takt garantiert ohne Wartezyklen
  - Kann gleichzeitig von DMA-Controller und S5 Engine benutzt werden
  - Syntax:

    ```
    __attribute__ ((section(".dataram.data")))
    ```
- On-Chip Static RAM: 256KB
  - Liefert Daten in $\approx$ 10 Takten (Wartezyklen möglich!)
  - Syntax:

    ```
    __attribute__ ((section(".sram.data")))
    ```
- Externes Double-Data-Rate Dynamic RAM (DDR-DRAM): bis zu 2 GB
  - Liefert Daten in 40-50 Takten (Wartezyklen möglich!)

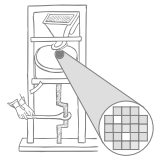
Speicheraufteilung

Alle Ressourcen in einem Adressraum zugänglich



A. Koch

Direct Memory Access (DMA)



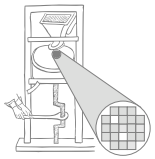
Bisher

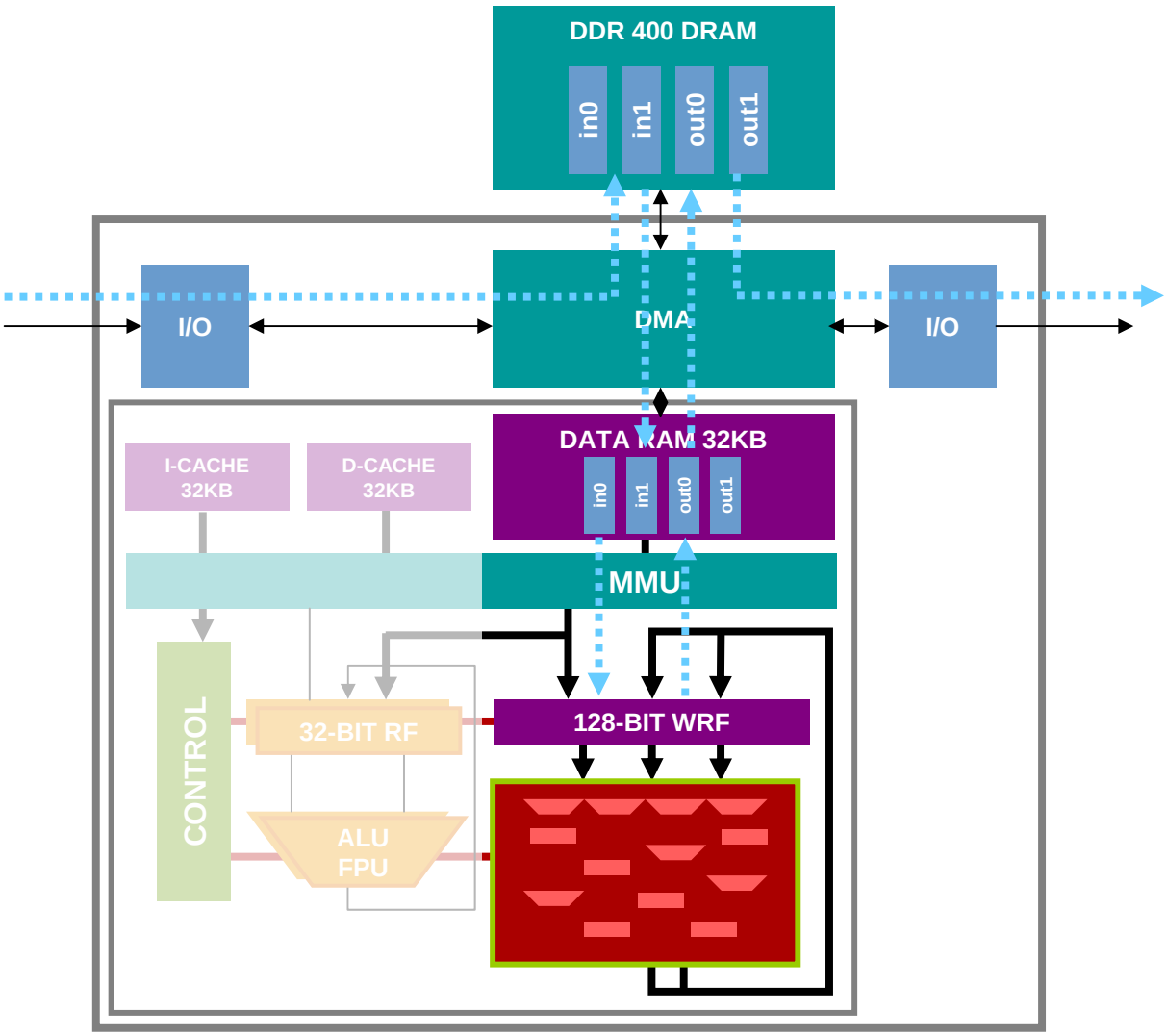
- Xtensa V Pipeline füttert ISEF mit Daten
- Explizite Übergabe von Registerwerten
- Explizite LOAD/STORE-Anweisungen

Alternative

- Datentransfers unabhängig von Xtensa V Pipeline
- Kann Daten-Cache umgehen
  - Sinnvoll bei Streaming
  - Hier keine temporale Lokalität
  - Cache-Effizienz eingeschränkt

Speicherzugriffsmechanismen





I/O → DDR-DRAM

DDR-DRAM → DPDRAM

DPDRAM → WRF

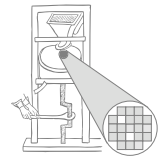
WRF → DPDRAM

DPDRAM → DDR-DRAM

DDR-DRAM → I/O

A. Koch

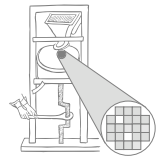
Beispiel für Programmierung von DMA-Transfers



A. Koch

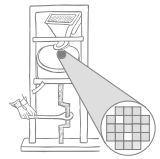
```
#include <sx-mmdma.h>

// Platz für zugewiesene DMA-Kanäle
sx_mmdma_chan *channelDDR2DPD; // von DDR-DRAM zu DPDRAM
sx_mmdma_chan *channelDPD2DDR; // von DPDRAM zu DDR-DRAM

// Steuerdaten für DMA-Transfer
sx_mmmdma_chan_config transferDDR2DPD;
sx_mmdma_chan_config transferDPD2DDR;

// Dummy-Daten
int indata[256], outdata[256];

// Zwischenpuffer
int inbuffer[256] __attribute__ ((section(".dataram.data")));
int outbuffer[256] __attribute__ ((section(".dataram.data")));

main() {
        ...
        // neue DMA-Kanäle anlegen (beliebiger HW-Kanal, 1 Transfer pro Kanal)
        transferDDR2DPD.chan_num = transferDPD2DDR.chan_num = -1;
        transferDDR2DPD.num_desc_buf = transferDPD2DDR.num_desc_buf = 1;
        sx_mmdma_chan_init(&transferDDR2DPD, &channelDDR2DPD)
        sx_mmdma_chan_init(&transferDPD2DDR, &channelDPD2DDR)

        // 1. DMA Operation von DDR-DRAM indata nach buffer in DPDRAM starten
        sx_mmdma_memcpy(channelDDR2DPD, inbuffer, indata, sizeof(indata), 0);

        // warten, bis Daten vollständig übertragen
        while (sx_mmdma_get_num_pending(channelDDR2DPD) > 1)
                ;

        // Daten von inbuffer nach outbuffer verarbeiten (z.B. mit ISEF)
        process(inbuffer, outbuffer)

        // 2. DMA-Operation von DPDRAM nach DDR-DRAM outdata starten
        sx_mmdma_memcpy(channelDPD2DDR, outdata, outbuffer, sizeof(outdata), 0);

        // warten, bis Daten vollständig übertragen
        while (sx_mmdma_get_num_pending(channelDPD2DDR) > 1)
                ;

        // Kanäle schliessen
        sx_mmdma_chan_close(channelDDR2DPD);
        sx_mmdma_chan_close(channelDPD2DDR);
        ...
}
```

Details der DMA-Benutzung



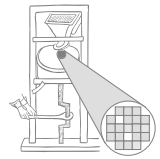
- Auch mehrere aufeinanderfolgende Transfers automatisch startbar
  - Dann **`num_desc_buf`** auf benötigte Anzahl erhöhen
  - Transfers wie üblich starten
  - **`sx_mmdma_get_num_pending`** liefert die Zahl der noch abzuarbeitenden Transfers des Kanals
- Eigenschaften je Kanal
  - Ursprungs- und Zielspeichertyp
    - DDR-DRAM, SRAM, DPDRAM, PCI, GIB, ...
  - DMA-Deskriptor
  - ➥Kann auch kompliziertere Transfers ausdrücken
- Eigenschaften je Transfer
  - Ursprungs- und Zieladresse
  - Transfergröße (in Bytes)

Idee Scatter/Gather-DMA



- Bisher lineare Datentransfers
  - Zusammenhängende Ursprungs- und Zielspeicherbereiche
- Scatter
  - Verteilen von Daten über unzusammenhängende Speicherbereiche
- Gather
  - Zusammenführen von unzusammenhängenden Daten zu einem Bereich

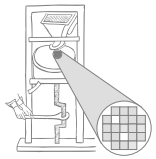
Scatter/Gather-DMA: Handhabung



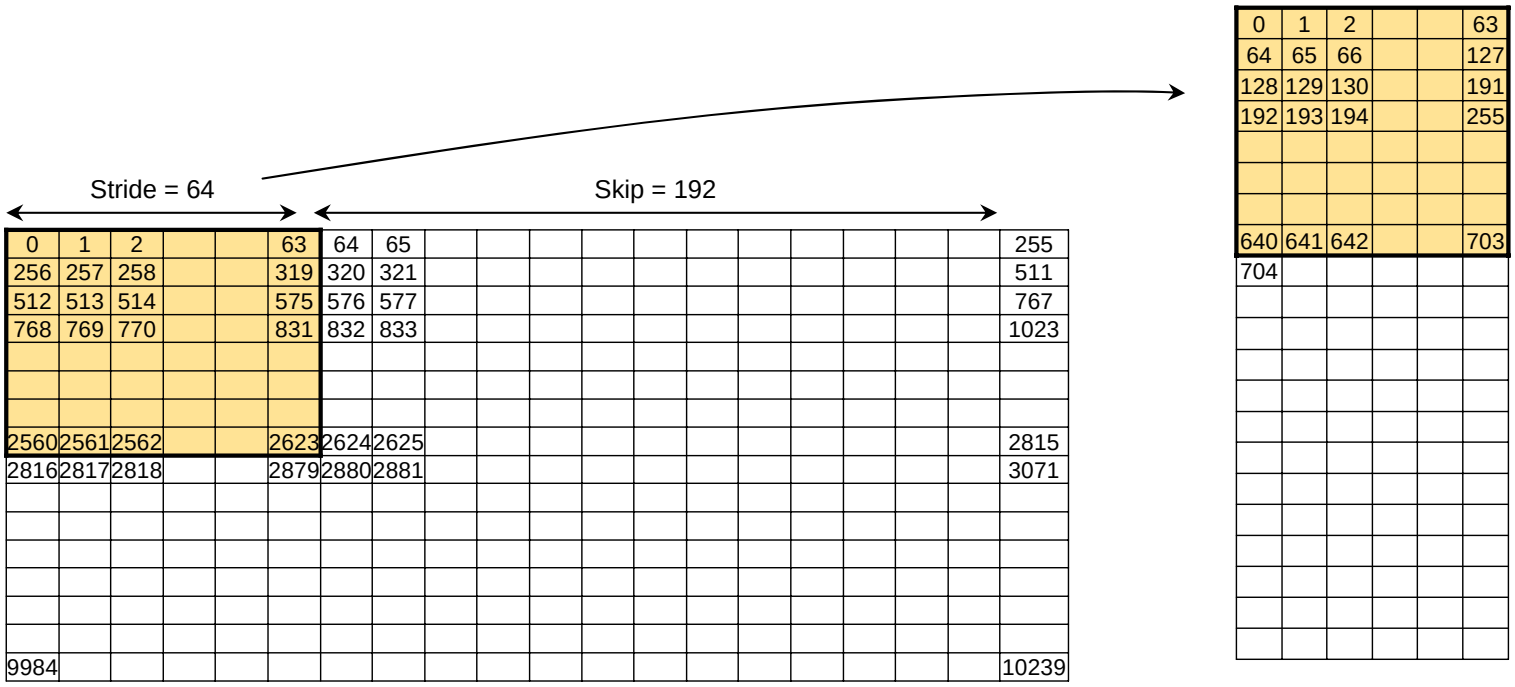
- Charakterisiert durch
  - Stride (Schrittweite)
    - Anzahl von aufeinanderfolgend übertragenen Bytes
  - Skip (Überspringen)
    - Anzahl von übersprungenen (nicht übertragenen) Bytes
- Angaben getrennt für Ursprungs- und Zielspeicherbereiche
- Transfer erfolgt durch abwechselnde Stride/Skip-Schritte
- Bis angegebene Gesamtanzahl von Bytes übertragen
- Lineare Transfers haben Skip = 0
  - Nichts wird übersprungen
  - Alle Bytes werden übertragen

Beispiel: Gather-DMA

Kopieren einer 11x64 Untermatrix aus einer 40x256 Matrix

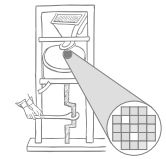


A. Koch

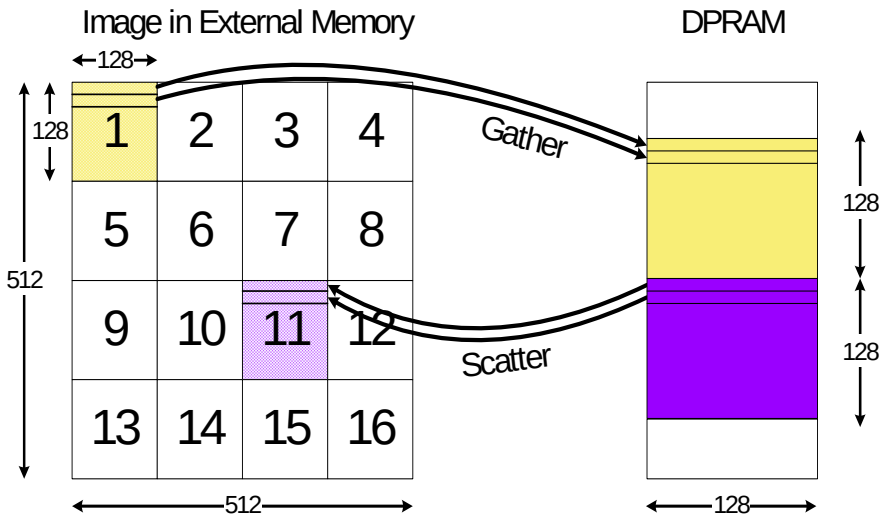


Beispiel: Scatter/Gather-DMA

Kopieren einer 128x128 Untermatrix einer 512x512 Matrix



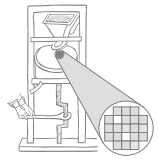
A. Koch



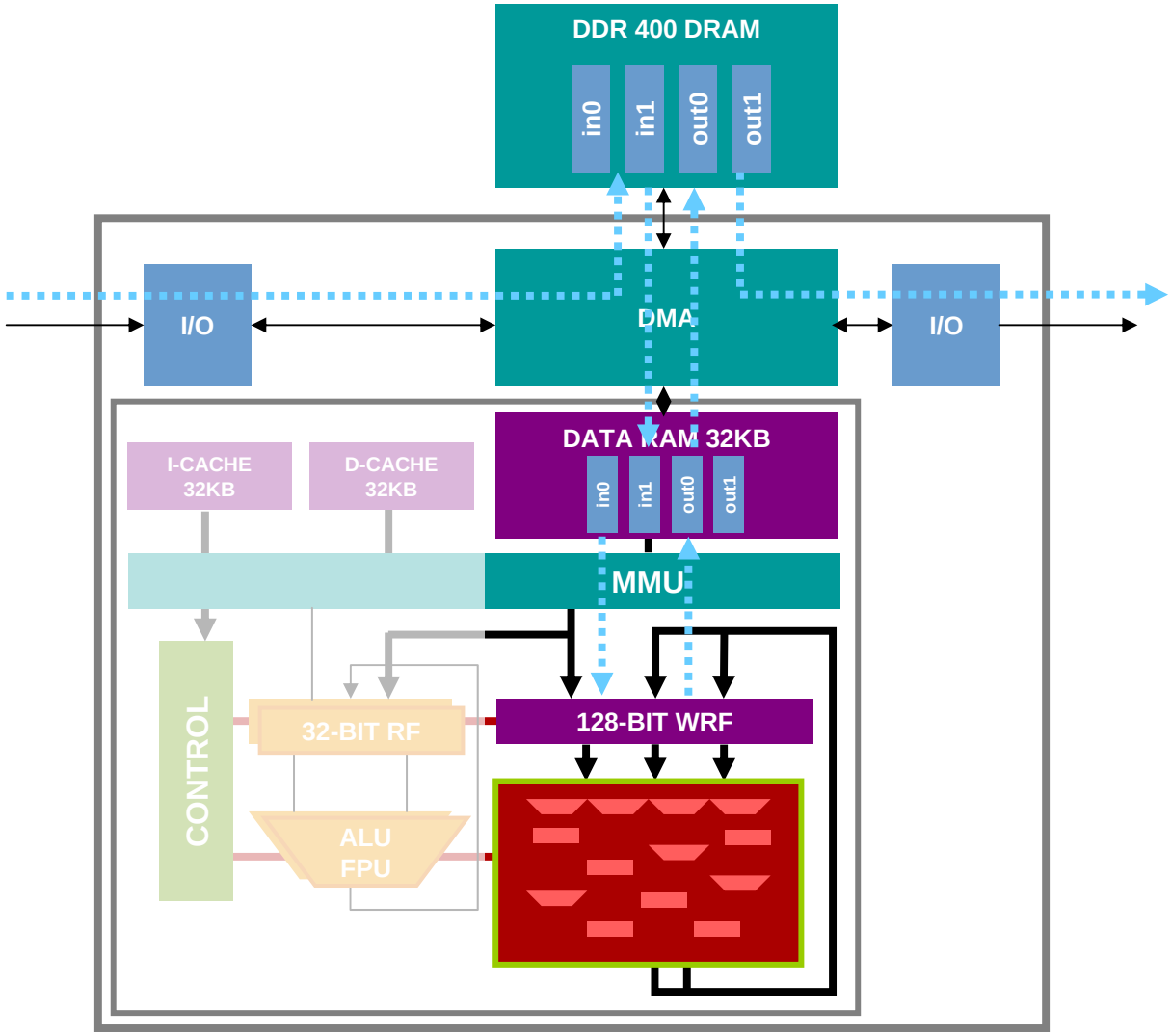

```
/* channel setup – Gather Function */
config.src_stride = 128; config.src_skip = 384;
sx_mmdma_chan_init (&config, &chID);

/* schedule memory-to-memory transfer */
sx_mmdma_memcpy (chID, dst, &src[0][0], 128*128, 0);

/* channel setup – Scatter Function */
config.dest_stride = 128;   config.dest_skip = 384;
sx_mmdma_chan_init (&config, &chID);

/* schedule memory-to-memory transfer */
sx_mmdma_memcpy (chID, &dst[256][256], src, 128*128, 0);
```

Interaktion von Speichersystemen

Caches und DMA

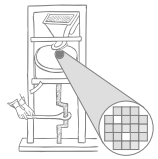


A. Koch



Cache-DMA Interaktion

Transfer nach DDR-DRAM: Problem?

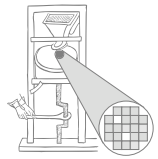


A. Koch

```
/* Initialization Code @ Reset */
for (i = 0; i < 1024; i++)
     buffer_In_DDR[i] = 0;

/* Generate some data and store in buffer_In_Dataram */
process (buffer_In_Dataram);

/* DMA out to buffer_In_DDR[ ] */
sx_mmdma_memcpy (chID, buffer_In_DDR, buffer_In_Dataram, 1024, 0);




/* Wait for transfer to complete */
while (sx_mmdma_get_num_pending (chID) > 0) ;

/* Print to view computed data */
for (i = 0; i < 1024; i++)
    printf (“Buffer[%d] = %d\n”,i, buffer_In_DDR[i])
```

Cache-DMA Interaktion

Transfer nach DDR-DRAM: Lösung

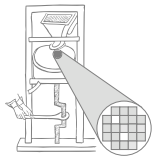


A. Koch

```
/* Initialization Code @ Reset */
for (i = 0; i < 1024; i++)
     buffer_In_DDR[i] = 0;

/* Generate some data and store in buffer_In_Dataram */
process (buffer_In_Dataram);

/* DMA out to buffer_In_DDR[ ] */
sx_mmdma_memcpy (chID, buffer_In_DDR, buffer_In_Dataram, 1024, 0);

/* Invalidate Data Cache  Why? */
sx_dcache_region_invalidate (buffer_In_DDR, 1024);

/* Wait for transfer to complete */
while (sx_mmdma_get_num_pending (chID) > 0) ;

/* Print to view computed data */
for (i = 0; i < 1024; i++)
    printf (“Buffer[%d] = %d\n”,i, buffer_In_DDR[i])
```

Cache-DMA Interaktion

Transfer nach DDR-DRAM: Lösung



A. Koch

```
/* Initialization Code @ Reset */
for (i = 0; i < 1024; i++)
     buffer_In_DDR[i] = 0;

/* Generate some data and store in buffer_In_Dataram */
process (buffer_In_Dataram);

/* DMA out to buffer_In_DDR[ ] */
sx_mmdma_memcpy (chID, buffer_In_DDR, buffer_In_Dataram, 1024, 0);

/* Invalidate Data Cache  Why? */
sx_dcache_region_invalidate (buffer_In_DDR, 1024);

/* Wait for transfer to complete */
while (sx_mmdma_get_num_pending (chID) > 0) ;

/* Print to view computed data */
for (i = 0; i < 1024; i++)
    printf ("Buffer[%d] = %d\n",i, buffer_In_DDR[i])
```

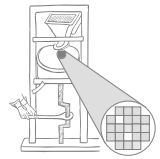
In sequence

1. Store to DDR
2. DMA to DDR
3. Load from DDR

Must invalidate data cache after DMA or load will get old data

Cache-DMA Interaktion

Transfer von DDR-DRAM: Problem?



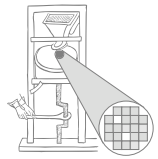
A. Koch

```
/* Write data to buffer */
for (i = 0; i < 1024; i++)
     buffer_In_DDR[i] = function (i);
```

```
/* DMA from to buffer_In_DDR[ ] */
sx_mmdma_memcpy (chID, buffer_In_DDR, buffer_In_Dataram, 1024, 0);

/* Wait for transfer to complete */
while (sx_mmdma_get_num_pending (chID) > 0) ;

/* Print to view data */
for (i = 0; i < 1024; i++)
    printf (“Buffer[%d] = %d\n”,i, buffer_In_Dataram[i])
```

Cache-DMA Interaktion

Transfer von DDR-DRAM: Lösung

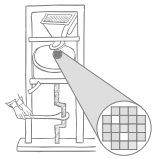


A. Koch

```
/* Write data to buffer */
for (i = 0; i < 1024; i++)
     buffer_In_DDR[i] = function (i);

/* Writeback Data Cache  Why? */
sx_dcache_region_writeback (buffer_In_DDR, 1024);

/* DMA from to buffer_In_DDR[ ] */
sx_mmdma_memcpy (chID, buffer_In_DDR, buffer_In_Dataram, 1024, 0);

/* Wait for transfer to complete */
while (sx_mmdma_get_num_pending (chID) > 0) ;

/* Print to view data */
for (i = 0; i < 1024; i++)
    printf (“Buffer[%d] = %d\n”,i, buffer_In_Dataram[i])
```

Cache-DMA Interaktion

Transfer von DDR-DRAM: Lösung



A. Koch

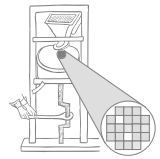
```
/* Write data to buffer */
for (i = 0; i < 1024; i++)
     buffer_In_DDR[i] = function (i);

/* Writeback Data Cache  Why? */
sx_dcache_region_writeback (buffer_In_DDR, 1024);

/* DMA from to buffer_In_DDR[ ] */
sx_mmdma_memcpy (chID, buffer_In_DDR, buffer_In_Dataram, 1024, 0);

/* Wait for transfer to complete */
while (sx_mmdma_get_num_pending (chID) > 0) ;

/* Print to view data */
for (i = 0; i < 1024; i++)
    printf (“Buffer[%d] = %d\n”,i, buffer_In_Dataram[i])
```

In sequence

1. Store to DDR
2. DMA from DDR

Must writeback data cache before DMA or DMA will get old data

Zusammenfassung



- Pipelining-Details
- Loop Unrolling
- Optimierung von EIs
- Konstantentabellen
- Rekonfiguration
- Heterogene Speicherressourcen
- DMA