

# TSEA44: Computer hardware – a system on a chip

Lecture 4: The lab system and JPEG encoding

## Practical issues

- Forming groups
  - Require pass on lab0
  - Send email to me (to get shared folder access)
  - Try to form groups of 3
    - Groups of 1 to 3 is expected, 3 is preferred, 2 is ok, 1 is accepted
    - See exam webpage of course for list of students

## Agenda

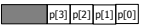

- Array/memory hints
- Cache in a system
  - The effect of cache in combination with accelerator
- Introduce JPEG encoding of images
  - DCT transform
  - Data reduction

## Some tips about arrays/memories

- FPGA memories can be created using
  - Flipflops; asynchronous read, synchronous write
  - Distributed using LUTs; asynchronous read, synchronous write, 16x1 each
  - BRAMs; synchronous read, synchronous write, 512x32, 1024x16 ....
- Memories can be designed
  - Using templates (BRAMs)
  - Inferred (distributed)

TSEA44: Computer hardware – a system on a chip 2018-11-14 5

### Some tips about arrays/memories

- Usually describe memory as arrays
- Two ways to describe arrays in SystemVerilog
  - Packed, e.g., logic [3:0] p;
    - Guaranteed to be continuous 
    - Typical for samples, values
  - Unpacked, e.g., logic u [3:0];
    - Support other data types 
    - Typical for multiple unit of same type

---

**li.u** LINKÖPING UNIVERSITY

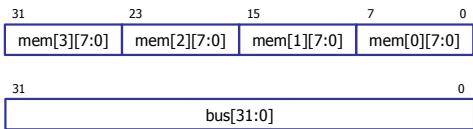
TSEA44: Computer hardware – a system on a chip 2018-11-14 7

### Packed arrays

```

wire [31:0] bus;           // a packed array
reg [3:0][7:0] mem;       // so is this
                           // both are continuous

assign bus = mem;
assign bus[31:16] = mem[3:2];
    
```




---

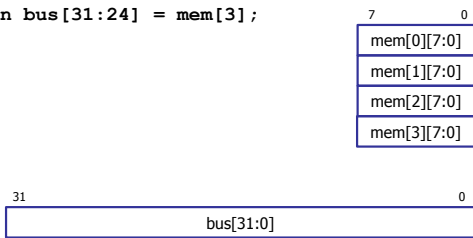
**li.u** LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 6

### Unpacked arrays

```

wire [31:0] bus;
reg [7:0] mem [0:3]; // a 4-byte memory
...
assign bus[31:24] = mem[3];
    
```

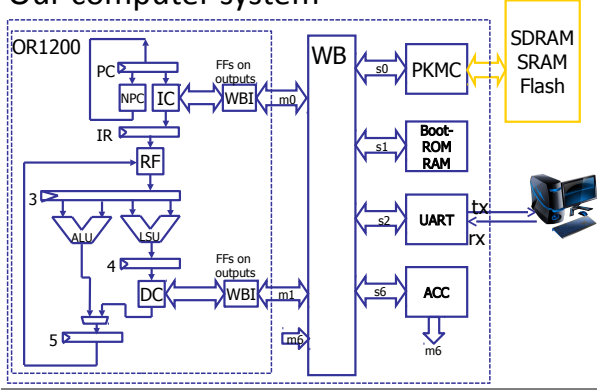



---

**li.u** LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 8

### Our computer system



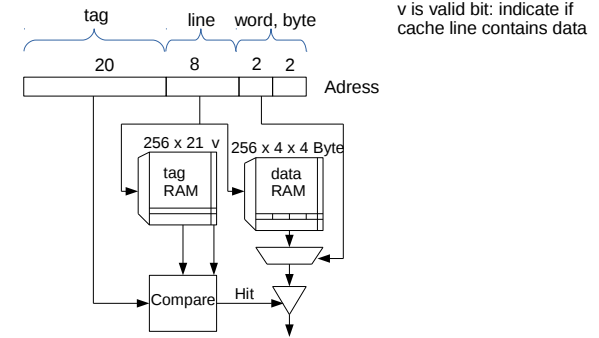

---

**li.u** LINKÖPING UNIVERSITY

### Caches

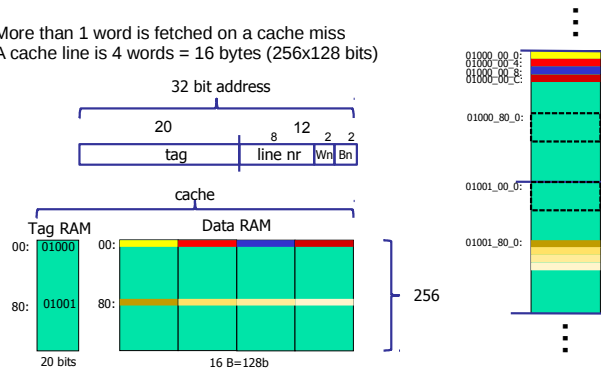
- Essential! Required to get good (close to 1) instructions per clock cycle (CPI)
- Expect to fetch 1 instruction each clock cycle
  - Internal (FPGA ROM/RAM) memory have a latency of 3 clockcycles
  - External (SRAM/SDRAM/FLASH) have a latency of 4 clockcycles
- **Size:** (depending on FPGA) there are up to 120 x 2KB block RAMs  
=> Select 8KB each for IC and DC
- **Type:** direct mapped (or set associative)

### 4 KB direct mapped cache

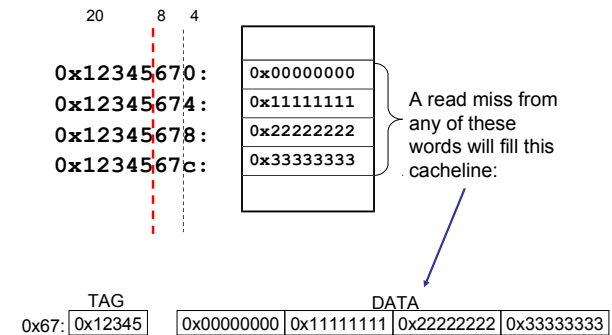


### 4 KB cache example (direct mapped)

More than 1 word is fetched on a cache miss  
A cache line is 4 words = 16 bytes (256x128 bits)



### A closer look at a cacheline



TSEA44: Computer hardware – a system on a chip 2018-11-14 13

### An IC cache miss

```

40000530: 9c 21 ff e4  l.addi r1,r1,0xffffffe4
40000534: d4 01 48 04  l.sw 0x4(r1),r9
40000538: d4 01 50 08  l.sw 0x8(r1),r10
    
```

LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 15

### Or1200 store buffer

- In a write-through data cache is every write equivalent to a cache miss!
- A store (write) buffer is placed between CPU and memory
- Writes are placed in a queue, so that the data cache is available on the next clock cycle

LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 14

### Cache policy

Cacheline = 4 words = 16 Bytes

#### Instruction cache

	hit	miss
read	read from cache	fill (replace) cacheline from memory

#### Data cache

	hit	miss
read	read from cache	fill (replace) cacheline from memory
write	write to cache write thru to memory	write to memory only

LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 16

### Watch out!

- Caches can be incoherent when using DMA
  - Write to memory not updating CPU caches
- Parts of memory should be non-cacheable
  - IC on for all addresses
  - DC on iff  $\text{addr}[31] == 0$   
 $\text{addr} < 0x8000\_0000$

LINKÖPING UNIVERSITY

## Accelerator interfacing

- Accelerator should implement functionality that is time-consuming to run on the CPU
- Interfacing the accelerator require additional data moves
- Simplest case (for the processor)
  - CPU send data to accelerator
  - CPU gets data from accelerator
    - Data available immediately, no waiting
    - Usually difficult to implement, processing takes time

## Accelerator interfacing, cont.

- Common for the accelerator to have large amount of data to receive, process, and return
- Simplest approach: Use CPU to feed accelerator with data
  - Mem->CPU                      Feed data to accelerator, uses CPU
  - CPU-> Accelerator
  - ...wait
  - Accelerator->CPU              Return data from accelerator, uses CPU
  - CPU -> Mem

## Accelerator interfacing, cont.

- More common case: Accelerator require some time to process data
  - CPU send data to accelerator
  - CPU waits for some time (N clock cycles)
    - No useful work performed by processor
  - CPU gets data from accelerator
  - Worse if time required to wait is unknown
    - Busy wait on the bus: Ask accelerator, but not get a respons for many clock cycles => Stalling CPU, locking bus

## Accelerator interfacing, cont.

- Want to reduce load on CPU: let the accelerator do the data moves by itself: DMA! (Direct Memory Access)
  - CPU setups DMA controller in accelerator (startadress, length)
  - Mem -> Accelerator    Feed data to accelerator, CPU do other things
  - ...processing
  - Accelerator->Mem    Return data from accelerator, CPU do other things

A drawback: Both accelerator and CPU compete for the bus  
Even worse if a number of accelerators work on data in sequence (Accelerator1 -> Accelerator2 ->...)

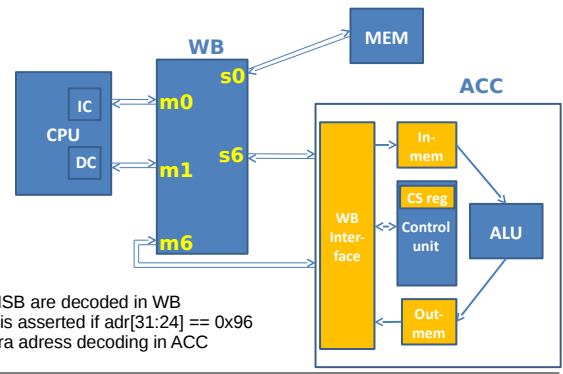
### Accelerator interfacing, cont.

- Stop communication between accelerators from going over the bus
    - Use special memories interconnecting only accelerators
    - Remove bus use (increase availability for the CPU)
    - The memories are unavailable to the CPU
- CPU->Accelerator (setup startaddress, length etc.)  
 Mem->Accelerator1  
 ... process in Accelerator1, store result in extra memory  
 ... process in Accelerator2, read input from extra memory  
 Accelerator2->Mem

### JPEG Introduction

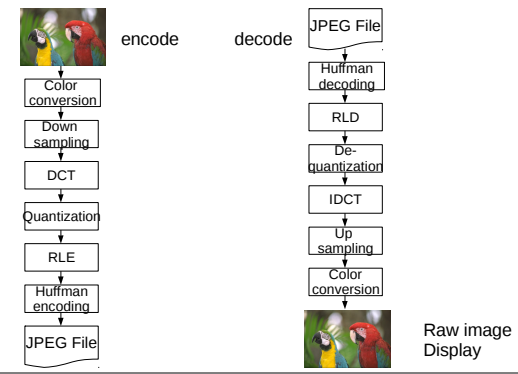
- Joint Photographers Expert Group
- Image compression standard defined by JPEG
  - Remove things that we cannot see
  - Decoded image is slightly different from original
    - Lossy compression

### Accelerator Control



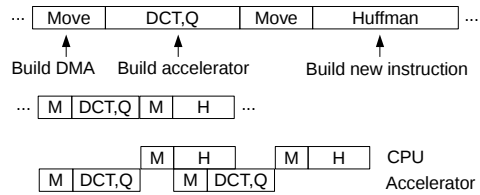
- 8 MSB are decoded in WB  
 stb is asserted if  $adr[31:24] == 0x96$   
 - Extra address decoding in ACC

### JPEG Encoding/decoding algorithm

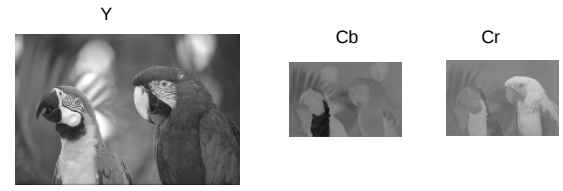


### Problem definition

- JPEG compression of testbild.raw 512x400 pixels
  - JPEG works on 8x8 blocks => 3200 blocks
  - Unaccelerated JPEG takes more than 32 000 000 clock cycles => 1 block takes more than 10 000 clock cycles!



### Resampling



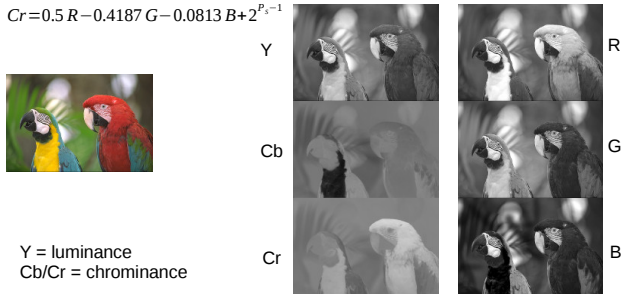
Data reduction 50%

### Color Conversion

$$Y = 0.299R + 0.587G + 0.144B$$

$$Cb = -0.1687R - 0.3313G + 0.5B + 2^{p_c-1}$$

$$Cr = 0.5R - 0.4187G - 0.0813B + 2^{p_c-1}$$



Y = luminance  
 Cb/Cr = chrominance

### 8-point 1-D DCT/IDCT

$$T(k) = c(k) \sum_{x=0}^7 v(x) \cos\left(\frac{(2x+1)k\pi}{16}\right), \quad k=0 \dots 7$$

$$v(x) = \sum_{k=0}^7 c(k) T(k) \cos\left(\frac{(2x+1)k\pi}{16}\right), \quad x=0 \dots 7$$

$$c(0) = \sqrt{\frac{1}{8}}$$

$$c(k) = \frac{1}{2}, \quad k \neq 0$$

$$C(x;k) = \cos\left(\frac{(2x+1)k\pi}{16}\right)$$

coord      freq

### 8x8-point 2-D DCT/IDCT

$$T(k,l) = c(k,l) \sum_{x=0}^7 \sum_{y=0}^7 v(x,y) C(y;l) C(x;k), \quad k,l=0..7$$

$$v(x,y) = \sum_{k=0}^7 \sum_{l=0}^7 c(k,l) T(k,l) C(y;l) C(x;k), \quad x,y=0..7$$

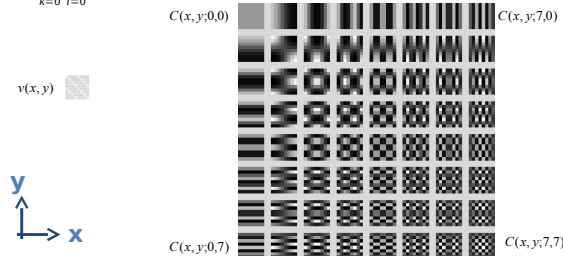
$$c(0,0) = \frac{1}{8} \quad k=l=0$$

$$c(k,l) = \frac{1}{4} \quad \text{else}$$

$$C(x;k) = \cos\left(\frac{(2x+1)k\pi}{16}\right)$$

### Meaning of the transform

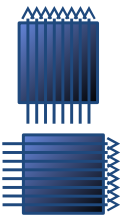
$$v(x,y) = \sum_{k=0}^7 \sum_{l=0}^7 T(k,l) c(k,l) C(y;l) C(x;k) = \sum_{k=0}^7 \sum_{l=0}^7 T(k,l) C(x,y;k,l)$$



### Simplifications

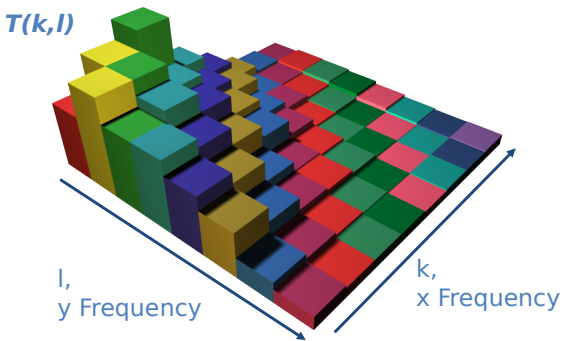
$$T(k,l) = c(k,l) \sum_{x=0}^7 \left\{ \sum_{y=0}^7 v(x,y) C(y;l) \right\} C(x;k)$$

$$= c(k,l) \sum_{x=0}^7 B(x,l) C(x;k)$$



2. 1-D DCT can be simplified for N=8

### Quantization





TSEA44: Computer hardware – a system on a chip 2018-11-14 33

### Data Reduction

- Transform, rounded division result  $Y_q = \text{round} ( \text{DCT}_2(Y)/Q_i )$

$$Y = \begin{bmatrix} 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 162 & 162 & 162 & 161 & 162 & 157 & 163 & 161 \\ 164 & 164 & 158 & 155 & 161 & 159 & 159 & 160 \\ 160 & 160 & 163 & 158 & 160 & 162 & 159 & 156 \\ 159 & 159 & 155 & 157 & 158 & 159 & 156 & 157 \end{bmatrix} \quad \text{DCT}_2(Y) = \begin{bmatrix} 259 & 5 & 3 & 0 & 0 & -1 & -5 & 6 \\ 8 & -1 & 1 & -5 & 2 & 3 & -4 & 3 \\ -5 & 0 & -2 & 2 & -1 & 0 & 2 & -2 \\ 2 & 1 & 2 & 1 & -1 & -1 & 0 & 1 \\ -1 & -1 & 0 & -1 & 2 & 1 & -1 & -1 \\ 1 & 0 & -2 & 0 & -2 & 1 & 2 & 1 \\ -2 & 0 & 3 & 2 & 2 & -2 & -1 & -1 \\ 1 & 0 & -2 & -2 & -1 & 2 & 1 & 1 \end{bmatrix}$$

$$Q_i = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} \quad Y_q = \begin{bmatrix} 16 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

**h.u.** LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 35

### Final modification

$k_3(k_1x + k_2y) = k_3k_1x + k_3k_2y$

precompute

**h.u.** LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 34

### Loefflers algorithm (fast DCT)

- 1-D 8-point DCT can be simplified

$$A[u] = c[u] \cdot \sum_{x=0}^7 a[x] \cos\left(\frac{2\pi}{32}(2x+1)u\right)$$

$\bigcirc \Rightarrow$  multiplication with  $\sqrt{2}$

$$x_{in} \begin{matrix} \text{---} \\ \text{---} \end{matrix} \begin{matrix} x_{in} + y_{in} \\ x_{in} - y_{in} \end{matrix}$$

$$\begin{matrix} x_{in} \\ y_{in} \end{matrix} \xrightarrow{cn} \begin{matrix} x_{out} \\ y_{out} \end{matrix} \Rightarrow \begin{cases} x_{out} = x_{in} \cdot \cos n\pi/16 + y_{in} \cdot \sin n\pi/16 \\ y_{out} = -x_{in} \cdot \sin n\pi/16 + y_{in} \cdot \cos n\pi/16 \end{cases}$$

**h.u.** LINKÖPING UNIVERSITY

TSEA44: Computer hardware – a system on a chip 2018-11-14 36

### RLE = run length encoding, Example:

Raw data: 0 0 0 1 1 1 1 0 0 0 0 1 0  
 Encoded as: 3:0, 4:1, 4:0, 1:1, 1:0

Alternative (if only zeroes are plentiful):  
 Raw data: 5 0 0 0 7 0 0 9 0 0 0 0 0 0  
 Encoded as: 5:3, 7:2, 9:DONE

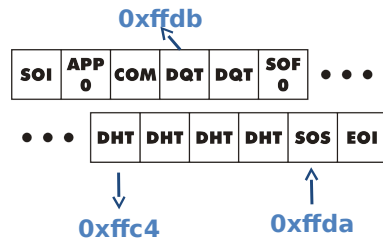
Special code

**h.u.** LINKÖPING UNIVERSITY



## JFIF Format

- JPEG File Interchange Format
  - Markers
  - Data



## Finally

- AC and DC values are treated differently
- Two Huffman LUTs are used
- DC
  - Differential, magnitude encoding, Huffman table lookup
- AC
  - As mentioned, raw bits left untouched, Huffman table lookup
- Example: value 04, raw bits 1100 => ....10111100....

in	code	length
0x00	1010	4
0x01	00	2
0x02	01	2
0x03	100	3
0x04	1011	4
0x05	11010	5
...	...	...

max length=16