Product Development - The Making of the Tiger Camera

by Mattias Ahnoff and Olve Maudal

TANDBERG Precision HD Camera er et 720p, 30fps, Pan-Tilt-Zoom kamera optimalisert for videokonferanse. Vårt Tiger-team i TANDBERG R&D på Lysaker utviklet dette kameraet fra scratch. Da det ble introdusert i 2006 hadde det gått bare 12 måneder fra ide til produksjon. Denne presentasjonen handler om video-prosessering og komprimering, CMOS sensorer, FPGA, DSP, C, Assembler og VHDL programmering. Dette er også en historie om hvordan en gruppe dyktige og enthusiastiske ingeniører jobber sammen for å lage produkter i verdensklasse.

TANDBERG

CHANGING THE WAY PEOPLE COMMUNICATE

Making them more productive by:

ACCELERATING DECISION MAKING

SCALING KNOWLEDGE

UNIFYING THE ORGANIZATION

PROMOTING WORK/LIFE BALANCE

TANDBERG

~~TANDBERG Television~~

~~Tandberg Data~~

~~Tandberg Storage~~

~~Tandberg ?~~

www.tandberg.com

1150 employees in 44 countries

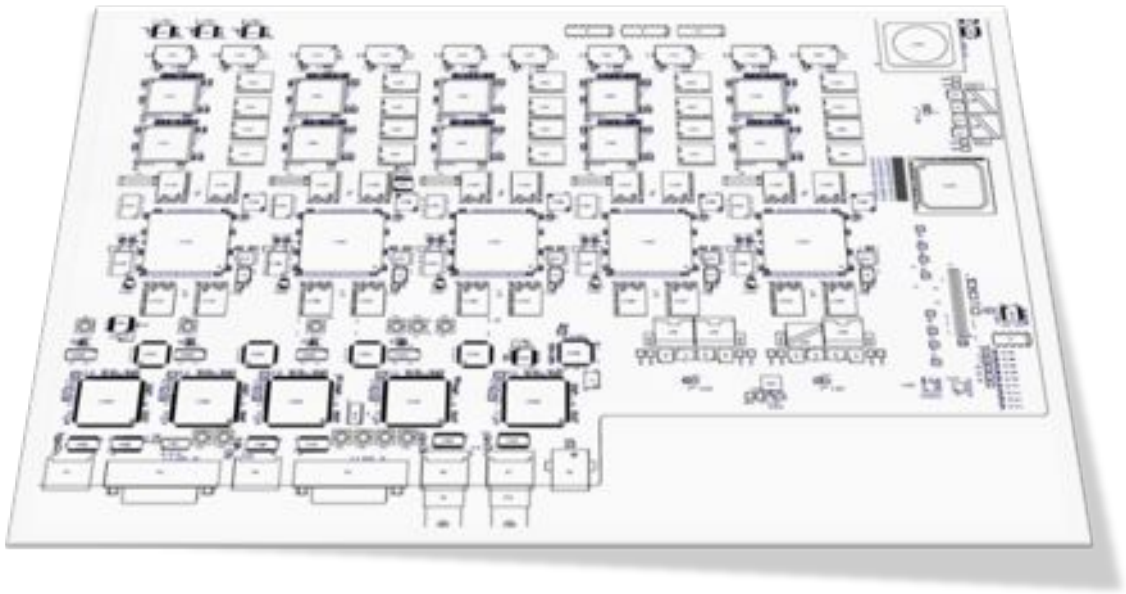
220 engineers in Norway

~160 programmers

C, C++, Java, C#, Python, VHDL

… but we also do

Electronics / Hardware

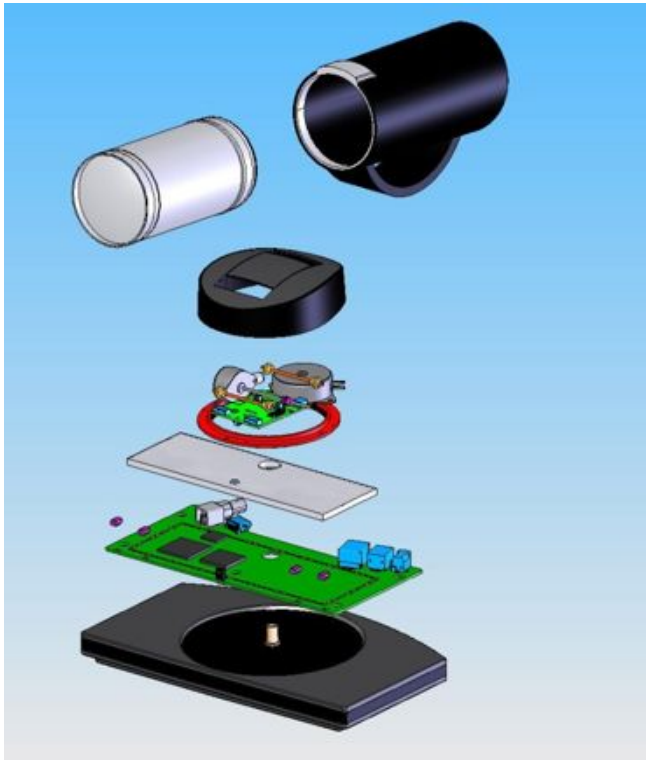


One of the most advanced boards in Norway:

- 6423 components

- 31392 pins

Mechanics



Industrial Design

1992

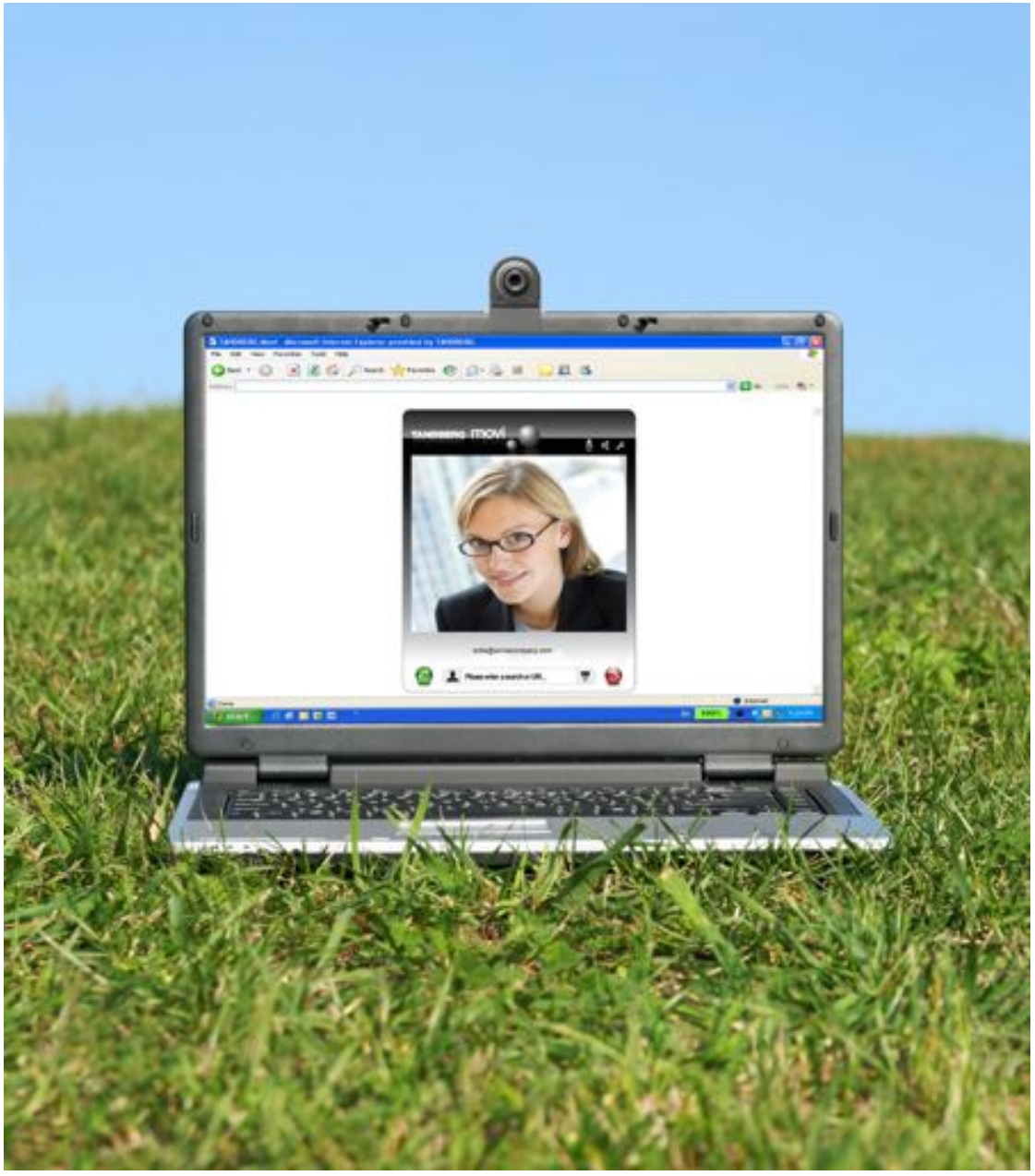
2007

Looking into

the Future

VHDL Ruby XML Dojo DSP JSP IPv6 XP hibernate C# RTP MPEG4 C Optics LCS Linux OOAD HTML UML Ubuntu CSS HDL Agile Maven Darwin WebServices GTK 65nm Windows V.35 Java FIPS DirectX ARM Struts gcc H264 LCD DSP DSPBios valgrind osgi Cocoa Objective-C realtime PowerPC Verilog vi Spring Qt Mac .Net gprof PCIe Outlook Plasma CMOS Imaging Ajax Multicore sRIO ISDN Emacs TCP/IP Lean C++ 8270 H323 Swing FPGA PSos TCL/TK Exchange Javascript Perl PNX1502 SIP MFC SQL Carbon FPGA AVR J2EE H263 EMC VoIP


Quarterly Report




TANDBERG 3G Gateway


TANDBERG MPS

Who uses videoconferencing technology?

Why do we need High-Definition?

WR, April 18, 2005

The Wainhouse Research Bulletin

ONLINE NEWS AND VIEWS ON VISUAL COLLABORATION AND RICH MEDIA COMMUNICATIONS

SPECIAL EDITION

LIFESIZE LAUNCHES 5-WAY SALVO

Andrew W. Davis, andrew.wd@wainhouse.com

With Focus on True High Definition, Start-Up Company Promises Users Quantum Leap in Videoconferencing Performance

What can a talented team experienced in the videoconferencing industry do with two years of development time, an unsurpassed network of personal contacts, and $30 million in venture capital? Quite a lot it turns out. LifeSize Communications has emerged from its two years of stealth operations – two years of secret, anti-marketing, no-leak operations that must be the envy of the CIA – with a suite of standards-based products centered on 1) high definition videoconferencing and 2) interoperability and compatibility with competitors' equipment already deployed in the field.

LifeSize's opening salvo covers FIVE separate products.

- **LifeSize Room** – a high definition (HD) video communications system for conference rooms. The system includes an HD codec; LifeSize-designed HD camera with 7X optical zoom; HD audio conference phone (also serves as an IP conference room phone) for microphone functions; and wireless remote control. The codec system connects to any display and delivers up to 1280x720 resolution at 30 frames/sec with a 1Mbps connection. At lower bandwidths, the codec maintains frame rate but drops the spatial resolution. The system includes an 8-way embedded MCU as a standard feature (only four video images are displayed at one time), multipoint bridging, spatial audio and dual streaming support for multimedia collaboration. The LifeSize Room system, including the LifeSize Phone has an MSRP of $11,999.

The LifeSize Room system with HD display

The innovative HD camera includes microphones in the base for direction finding only, providing spatial cues to the far end.

The LifeSize codec can be mounted horizontally or vertically

The complete room system

- **LifeSize Exec** – a high definition executive video communications system designed for use in offices or small conference rooms with 1-4 people. This is a fully integrated unit including a 17", 16:9 monitor display; high definition camera with 70 degree fixed wide angle lens; built-in HD audio conference phone; headset jack; WiFi interface and wireless remote. The Exec connects to any display and delivers up to 1280x720 resolution, 30 frames per second and operates over 6MBps to 1Mbps. The Exec system includes as standard the same 8-way embedded MCU and dual streaming support for multimedia collaboration. List Price: $7,999.

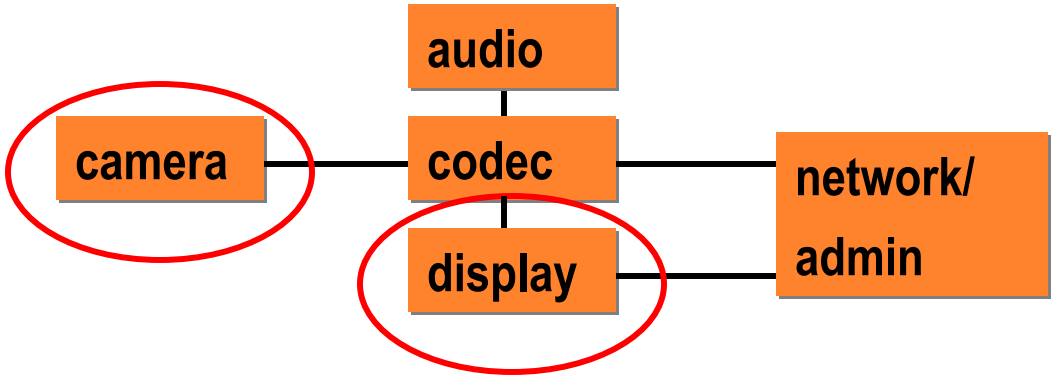
LifeSize units include a 16x9 1280x720 pixel display, built-in wide-band conference phone, 8-way audio-video bridge, wifi connection, and a headphone jack. The device folds flat for carrying

The LifeSize Phone is fully integrated with the room system, but also serves as a standalone IP or PSTN conference room phone.

- **LifeSize Phone** – a high definition audio conference phone with innovative, circular design and 16 microphones. As noted above, the Phone is a standard part of the video system, providing the required table-top microphone inputs; in this configuration the phone can be used to dial video calls as well. In addition, the Phone can be used stand-alone as an IP conference room phone. The Phone includes many engineering innovations. The Circular Microphone Array technology uses 16 separate microphones and beam forming technology; LifeSize claims higher directivity and twice the room coverage of any existing solution. The device uses the MPEG-4 AAC algorithm for up to 22 kHz audio depending on the connectivity type. The company claims that its three inch/ 8cm custom speaker driver and tuned-port result in 10x lower distortion than any other conference phone and superior bass without a subwoofer. List Price: $1,199 (IP) and $1,399 (PSTN).

Why design our own camera?

- We did not find a camera that met our requirements for a high end camera at a price good enough
- A camera optimized for videoconferencing
- Expand core competence from video compression to camera and display technology to improve the ”Visual Communication” experience



- July 2005. Project starts.
- Code name Tiger.
- High risk projec
- High attention

Target goals

- Improved resolution
- Improved optical performance, better sharpness and less distortion
- Improved noise reduction in low light environment
- Image data output is optimized for our codec h/w and for our compression algorithm
- One camera worldwide with no 50/60Hz flicker
- Improved colors and dynamic range
- Better price/performance ratio

Deciding which H/W to use

- Per-pixel processing very well suited for FPGA
- Control code in DSP or soft core in FPGA

But…

- Development time
- Rescources
- Rapid prototyping => DSP

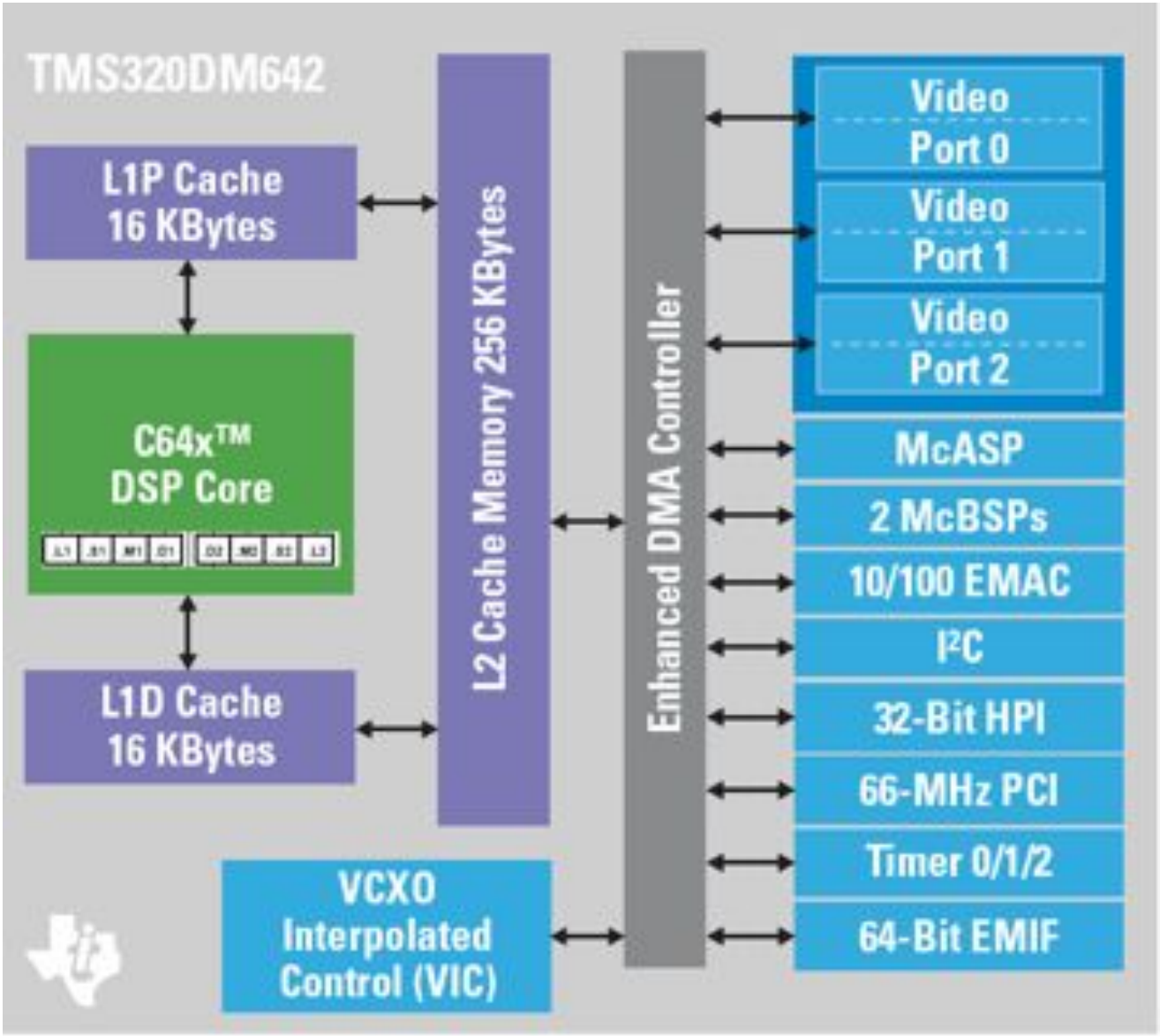
So we decided on DSP + FPGA

- Minimizing risk
- Optimizing time to market

Processing power / data flow

- Needed to decide how much processing power to use
  - Fast dummy implementations for benchmarking purposes
  - But we did not yet know which algos to use
- How to connect FPGA and DSP
  - Where to split in terms of image pipeline
  - Limitations on what can be done where
- Optimizing for flexibility

DSP


TMS320DM642
L1P Cache
16 KBytes
C64x™
DSP Core
L1D Cache
16 KBytes
L2 Cache Memory 256 KBytes
Enhanced DMA Controller
VCXO
Interpolated
Control (VIC)
Video
Port 0
Video
Port 1
Video
Port 2
McASP
2 McBSPs
10/100 EMAC
I²C
32-Bit HPI
66-MHz PCI
Timer 0/1/2
64-Bit EMIF

FPGA

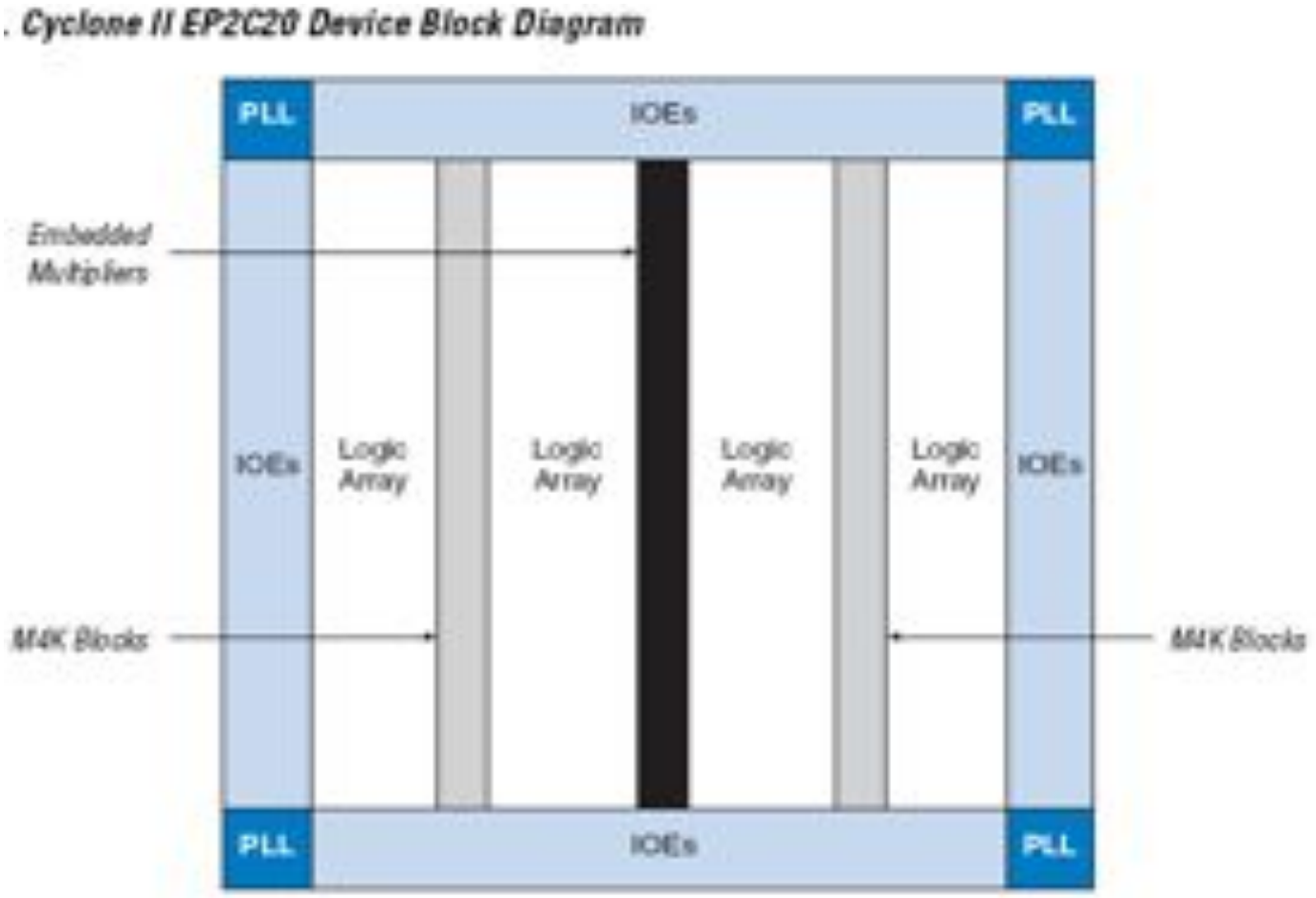
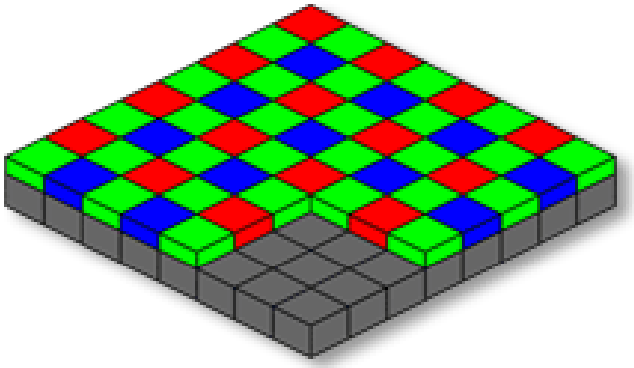
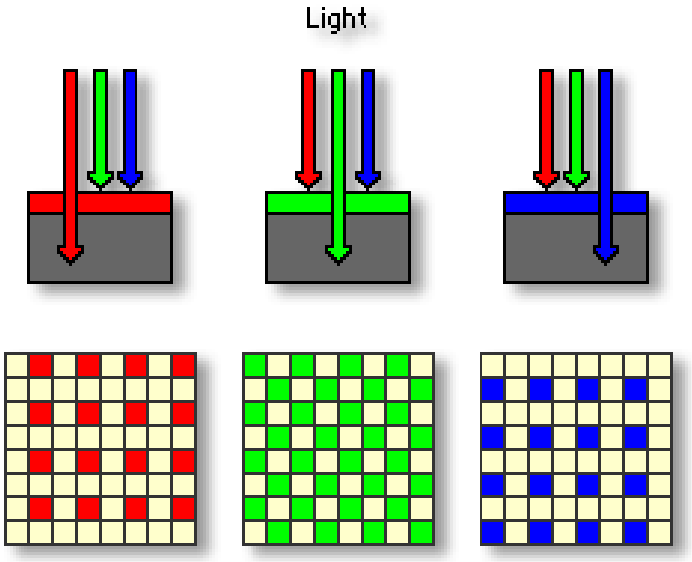

. Cyclone II EP2C20 Device Block Diagram
PLL
IOEs
PLL
Embedded Multipliers
IOEs
Logic Array
Logic Array
Logic Array
Logic Array
IOEs
M4K Blocks
M4K Blocks
PLL
IOEs
PLL

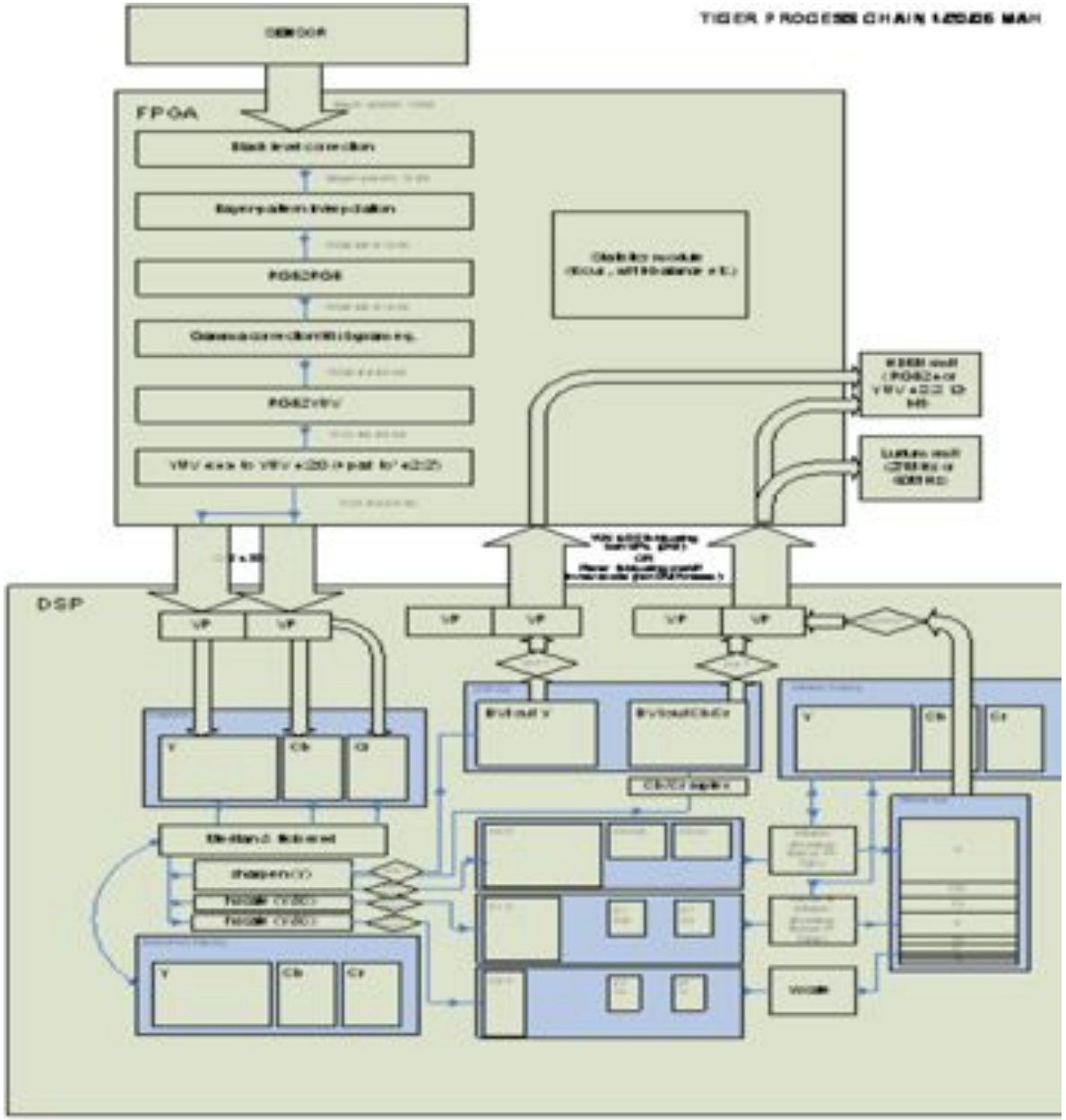
Image Sensor

- 1/3” 1.3 Mpixel CMOS (720p60)



Color Filter Array Sensor

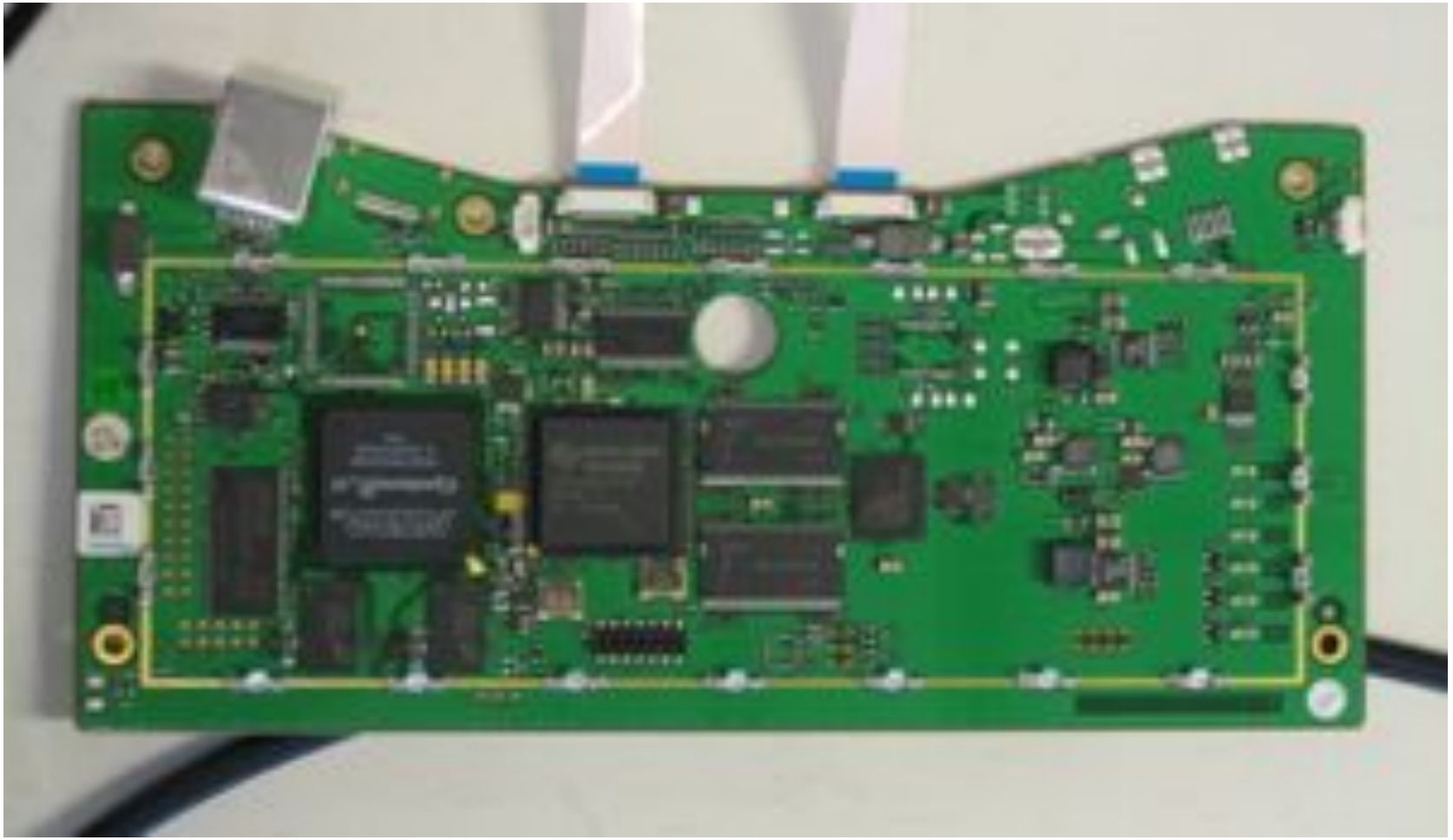



FPGA
DSP
VP
VP
VP
VP
VP
VP
Y
Cb
Cr
Y
Cb
Cr
Y
Cb
Cr

Lots of s/w inside the camera

- Image pipeline, AE, AF, sensor control
- Motor control (PTZ)
- Interaction with codec (VISCA, LVDS, IR remote)
- Code for debug (direct or telnet via codec)
- Calibration during production
- Tests during production
- Boot s/w
- Automatic s/w upgrades
- s/w to support codec compression

Precision HD mainboard, Dec 20, 2005



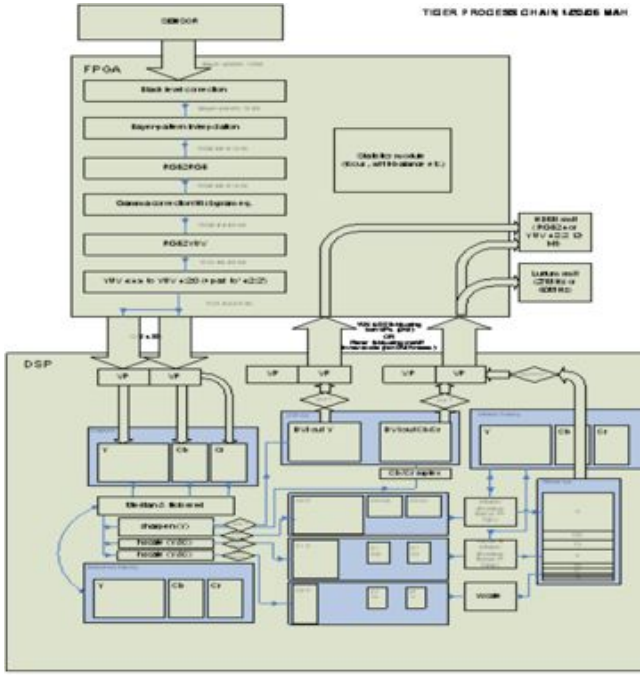
FPGA, DSP, RAM, connection to sensor. No fan.

Jan 4, 2006

“ Klarte å få igjennom de første bildene fra Tiger i går kveld…

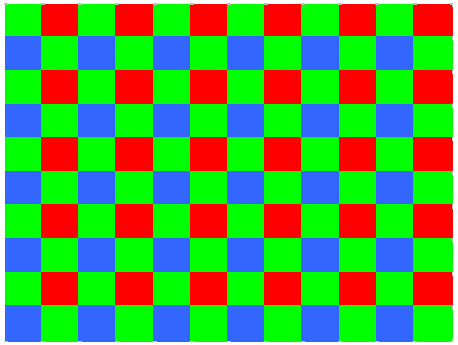
trenger en del justeringer før ting ser bra ut…”

Image pipeline examples



Demosaicking

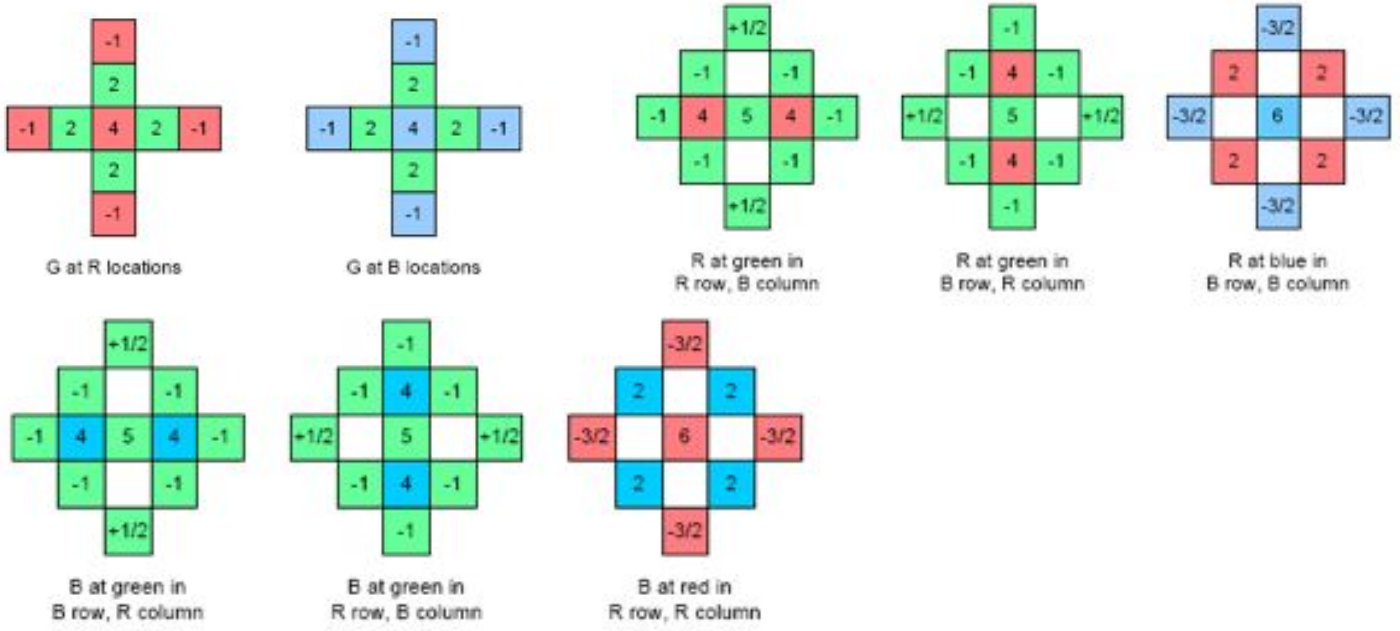
- For a 1-chip solution, there is only one of R, G, and B in each pixel position.
- Bayer-pattern most common configuration
- 50% G, 25% R and 25% B



Demosaicking

Example:

HIGH-QUALITY LINEAR INTERPOLATIONFOR DEMOSAICING OF BAYER-PATTERNED COLOR IMAGES *Henrique S. Malvar, Li-wei He, and Ross Cutler,* Microsoft Research



We use a different scheme (not published)

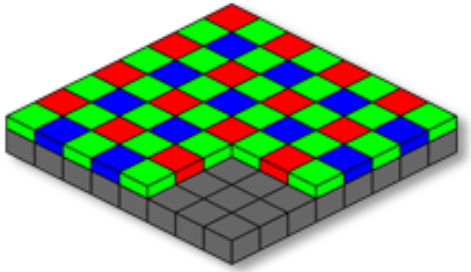
Figure 3. Demosaicing results for various interpolation algorithms. From left to right: top row: original, bilinear, Freeman [2],[5]; middle row: Laroche-Prescott [2],[6], Hamilton-Adams [6],[7], Chang et. al. [8]; bottom row: Pei-Tam [4], Kimmel [3], and our proposed method.

REFERENCE: HIGH-QUALITY LINEAR INTERPOLATIONFOR DEMOSAICING OF BAYER-PATTERNEDCOLOR IMAGES

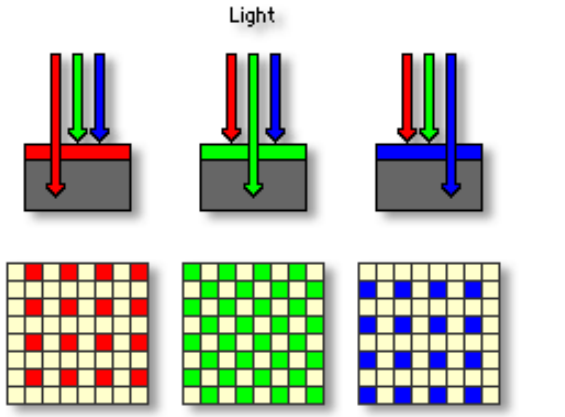
Henrique S. Malvar, Li-wei He, and Ross Cutler, Microsoft Research

Colour correction

- Why colour correction?
  - CFA

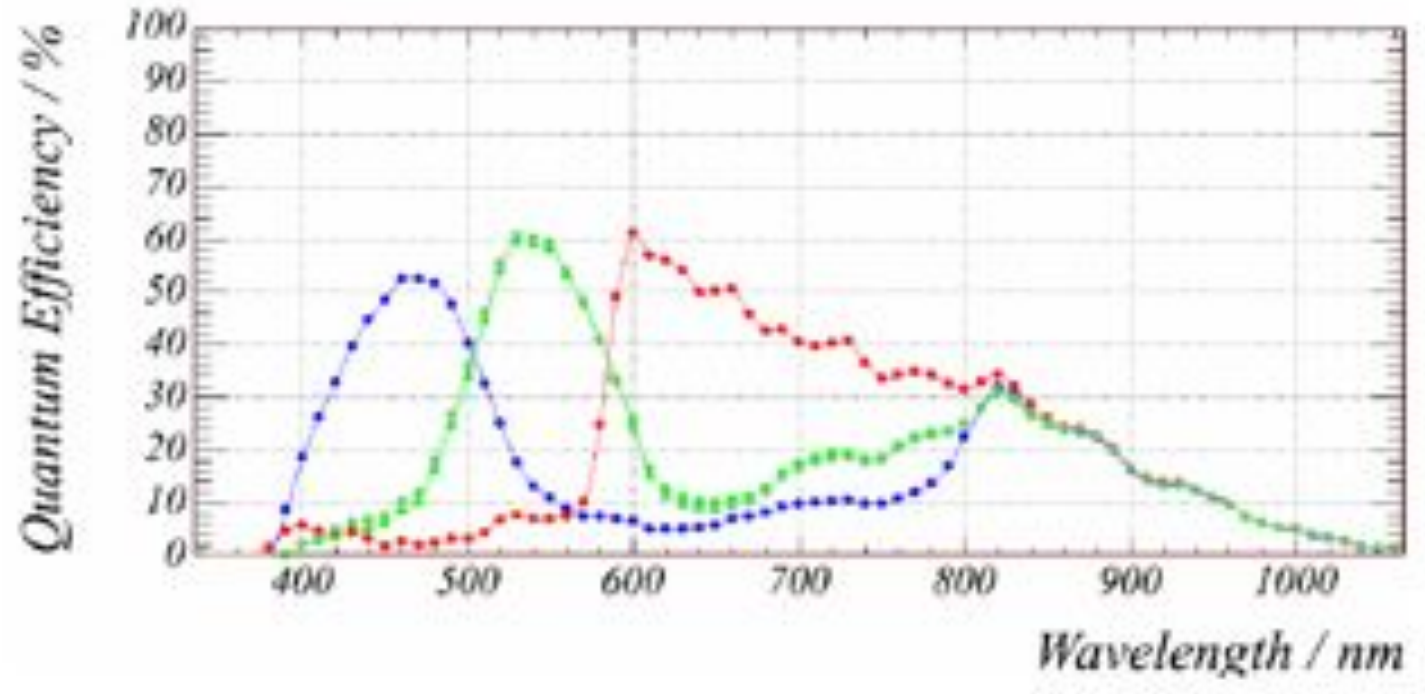


Color Filter Array Sensor



Colour correction

- CFA overlap



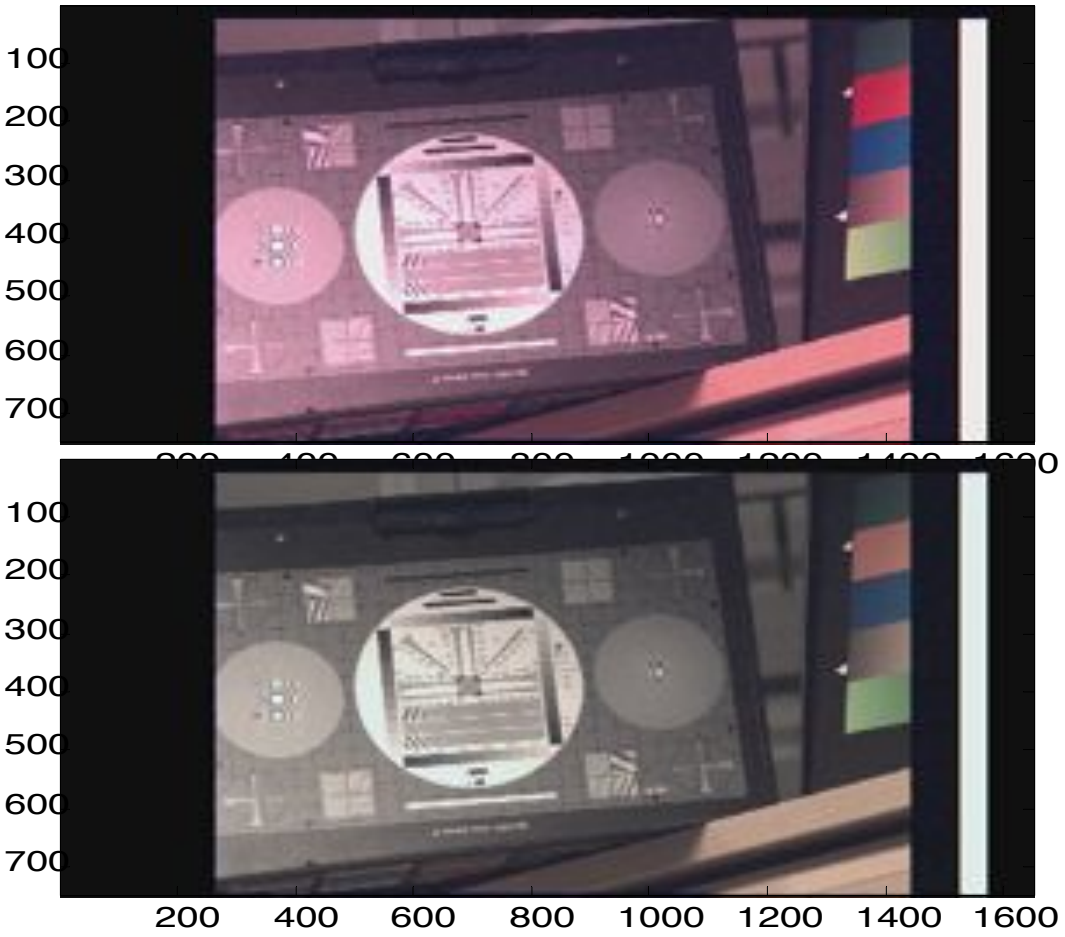
White balance

Make white look white

- Gray world assumption: sumRed = sumGreen = sumBlue
  - Works surprisingly often, but not always
- Highlight = white
  - difficult in reallity because highlight likely to be saturated
- Manual calibration
  - Difficult in our application

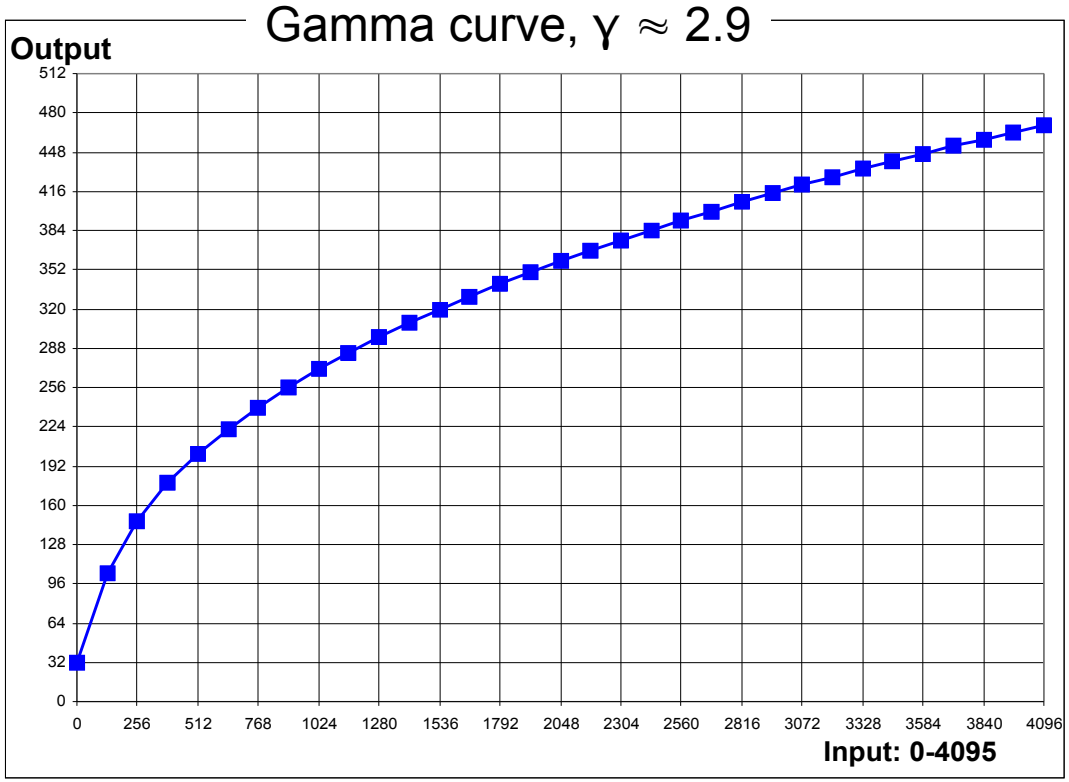
Most WB algorithms are based on Gray World Assumption, plus some magic to detect and deal with those situation where it fails

Colour correction and white balance

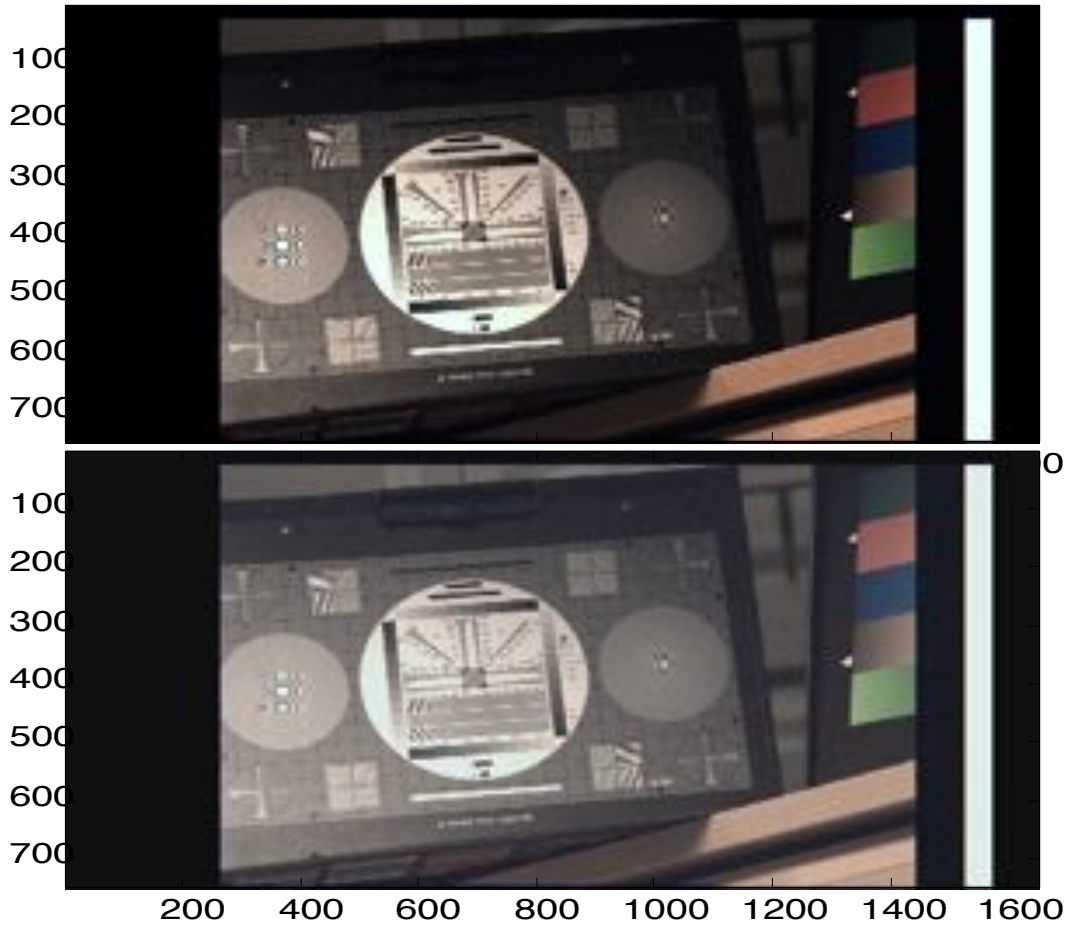


Gamma correction (1/2)

- Enhance low- to mid-range
- 8 tables stored as LUT in FPGA (32 entries each)
- Use linear interpolation
- We use $F(x) = a(1-x^{0.3})x + x^{(0.3a+3/a)}$



Gamma correction (2/2)



RGB2YUV

- RGB 4:4:4 to YUV 4:2:2
- Reduce bandwidth FPGA → DSP

RGB / YUV decomposition

RGB

YUV

RGB / YUV decomposition

RG

+

B

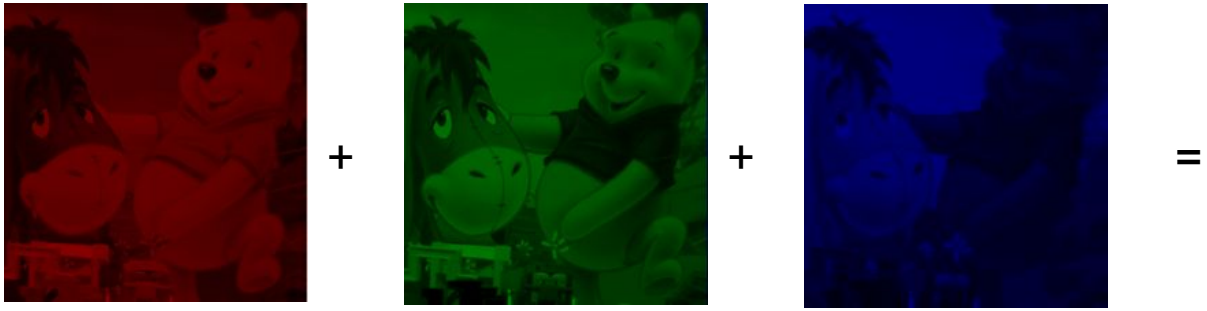
YU

+

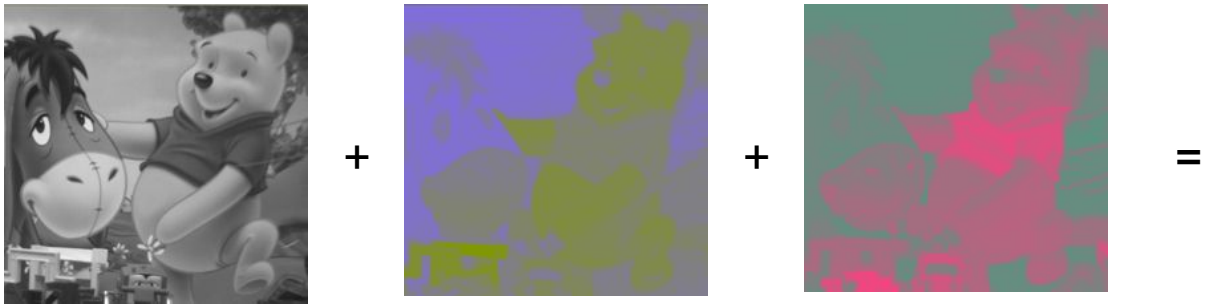
V

RGB / YUV decomposition

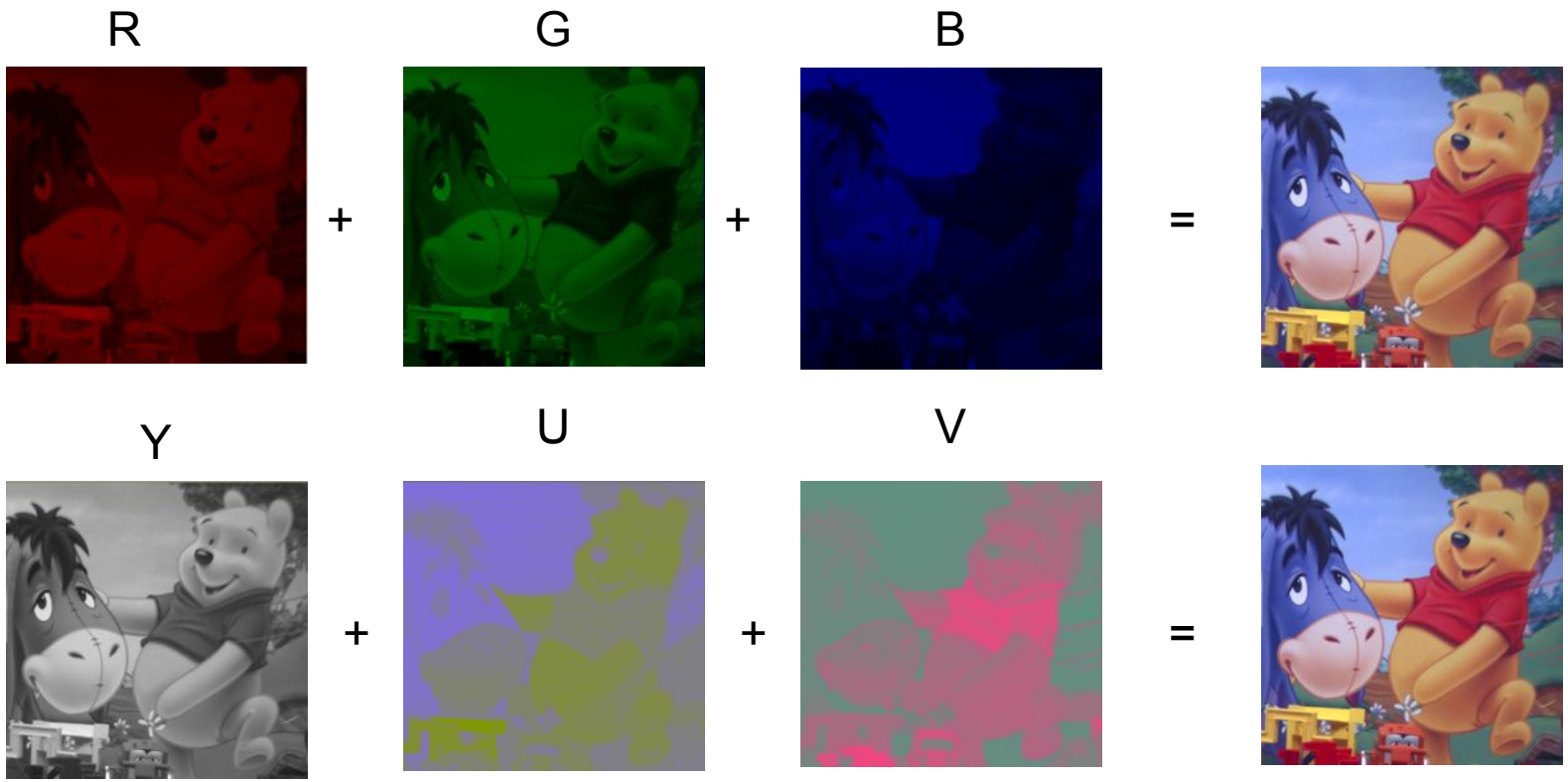
R G B



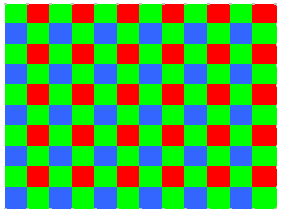
Y U V



RGB / YUV decomposition



CFA



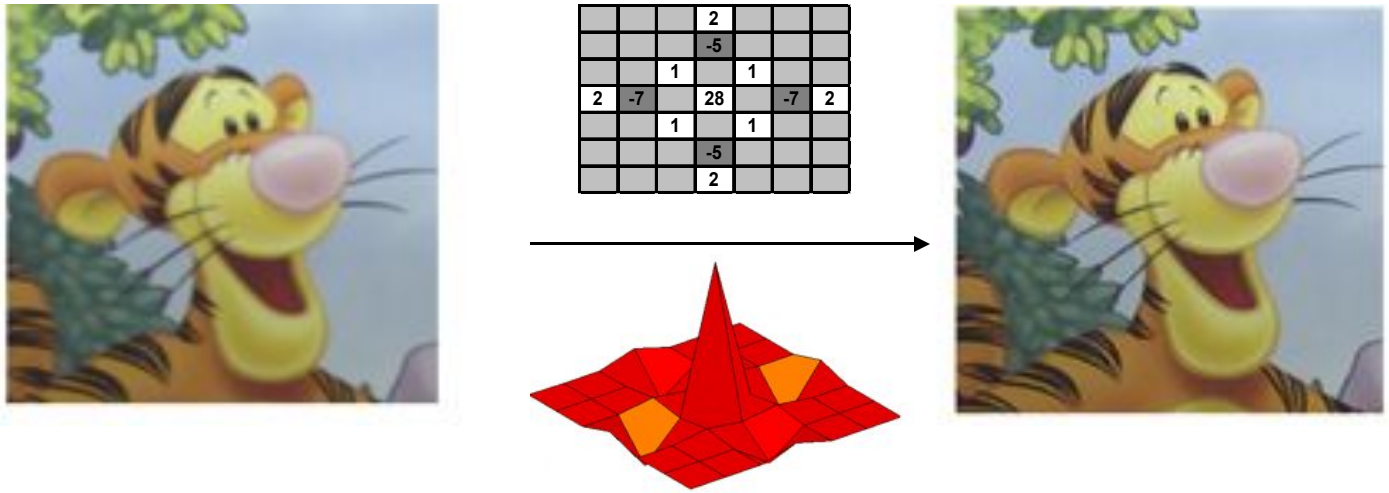
- More G- than R-, B-pixels because G is largest contributor to Y:

$$
\begin{aligned}
Y &= 0.30\,R + 0.58\,G + 0.11\,B \\
U &= -0.17\,R - 0.34\,G + 0.51\,B \\
V &= 0.51\,R - 0.43\,G - 0.08\,B
\end{aligned}
$$

Sharpen Filter / Edge Enhancement

Sharpen filter:

- **Boost high frequencies (e.g. by using 2D FIR filter) to make image look sharper.**



Edge enhancement:

- **Try to detect edges and apply the sharpen filter selectively.**

Color suppression

- Moiré artifacts occur as image is not lowpass filtered before it's being sampled.
- Demosaicing artifacts.

Try to detect where artifacts are likely to occur and suppress color (i.e. make image more black&white) there.

Color suppression (cont'd)

Before

After

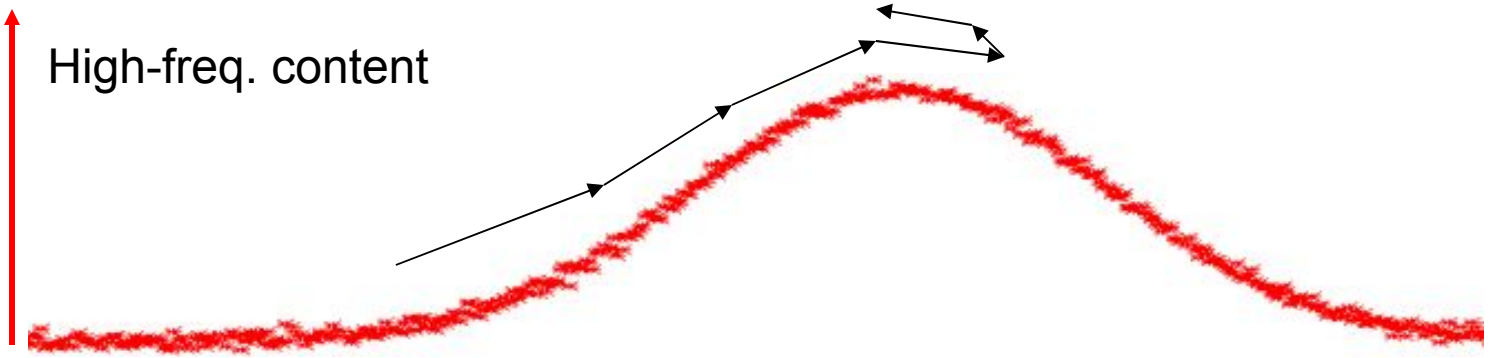
This is our proprietary color suppression algorithm.

Auto-exposure

- Can adjust analog/digital gain, Iris and Integration Time
  - Step size vs Speed
  - Limitations 50Hz/60Hz
  - Not too fast (be nice to H.264 encoder)
  - Can make ”backlight compensation” manually
- Based on highlight and average measurements

Auto focus

- Analyze image to determine sharpness (for instance by computing energy of high-pass filtered image)
- Iterate until max sharpness => in focus



Auto focus considerations

- Focus curve might have many local maxima
- Focus on a region of the image? If so, which?
- When to re-focus? Keep in mind H.264 encode.
- Roubustness against noise, movements etc.
- Focus tracking during zoom?

- Ideally: fast, accurate, roubust and intuitive.

Image resizing

- Resizing in one direction at a time:

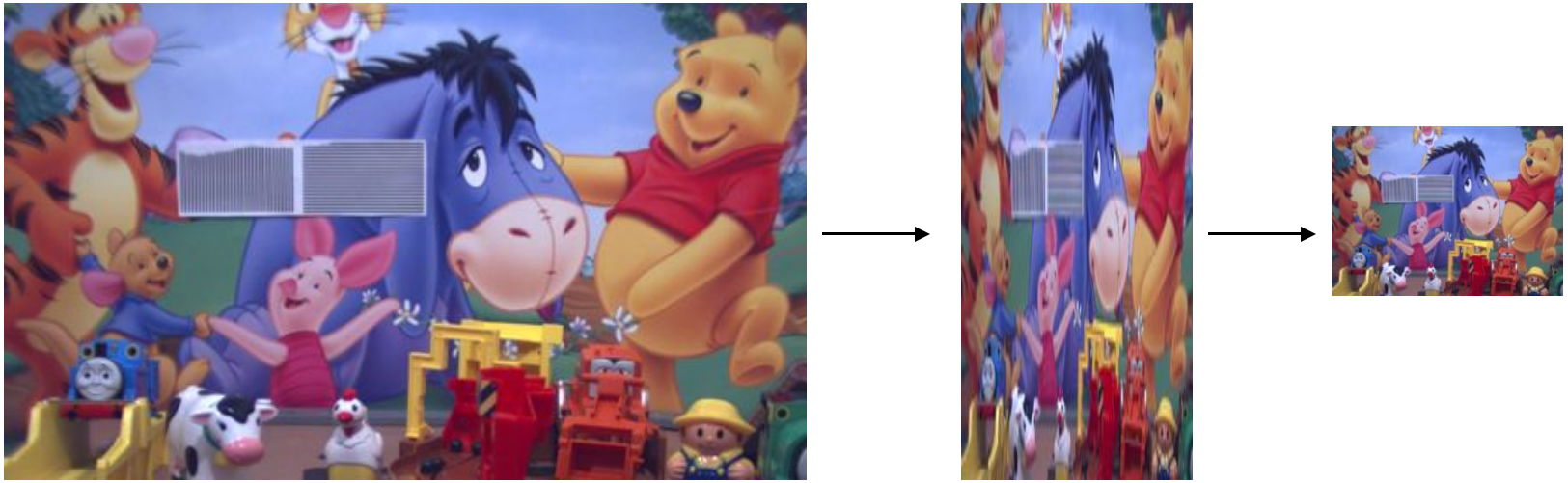
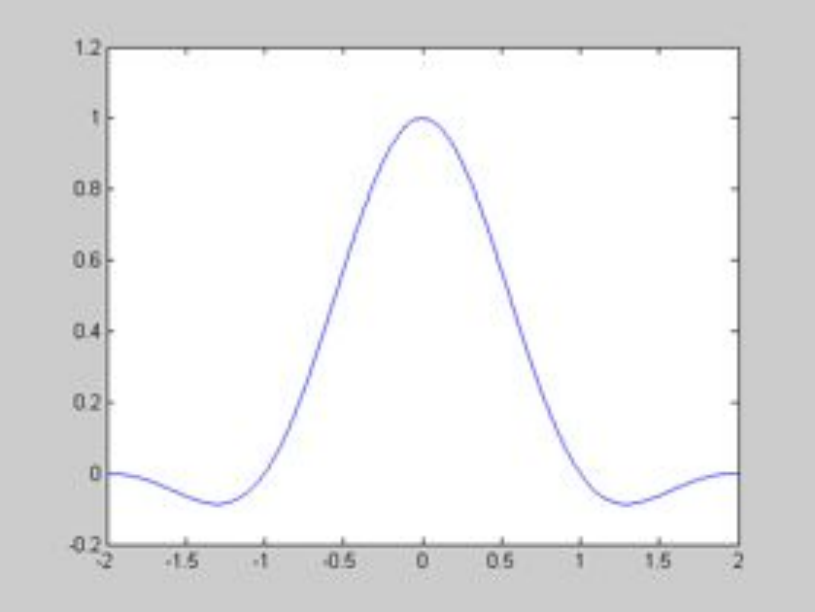


Image resizing

- Lanczos resize kernel



$$L(x) = \begin{cases} \frac{2\sin(\pi x)\sin\left(\frac{\pi}{2}x\right)}{\pi^2 x^2} & -2 < x < 2, x \neq 0 \\ 1 & x = 0 \\ 0 & otherwise \end{cases}$$

Image resizing

Typically need several phases. Example: d/s by 5/4

```
in:   y   c   y       y   c   y       y   c   y       y   c   y       y   c   y       y
out:  y    c    y         y    c    y         y    c    y         y    c    y         y
```

Four phases:

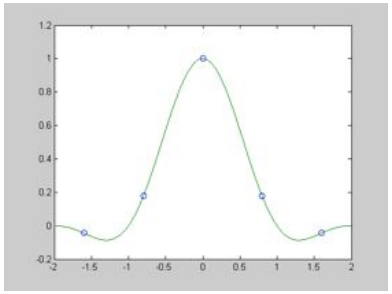
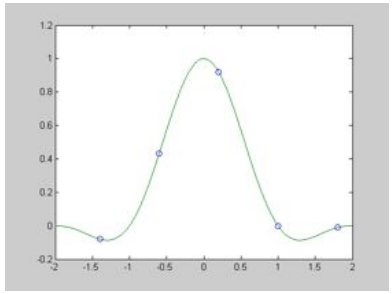
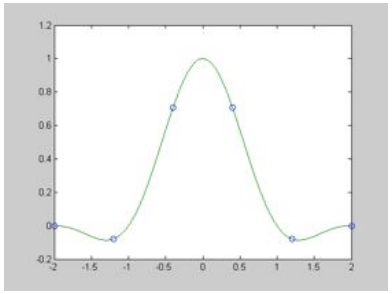
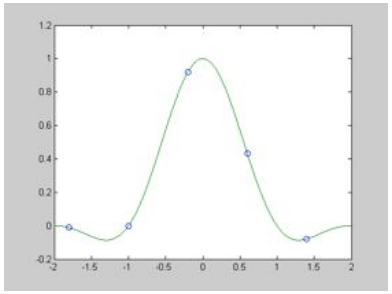
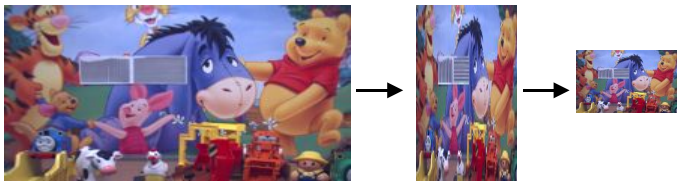
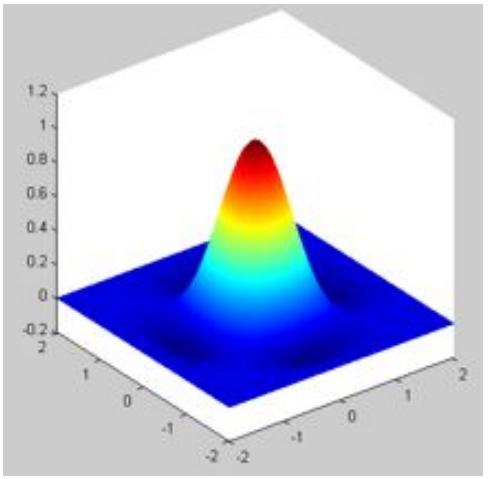
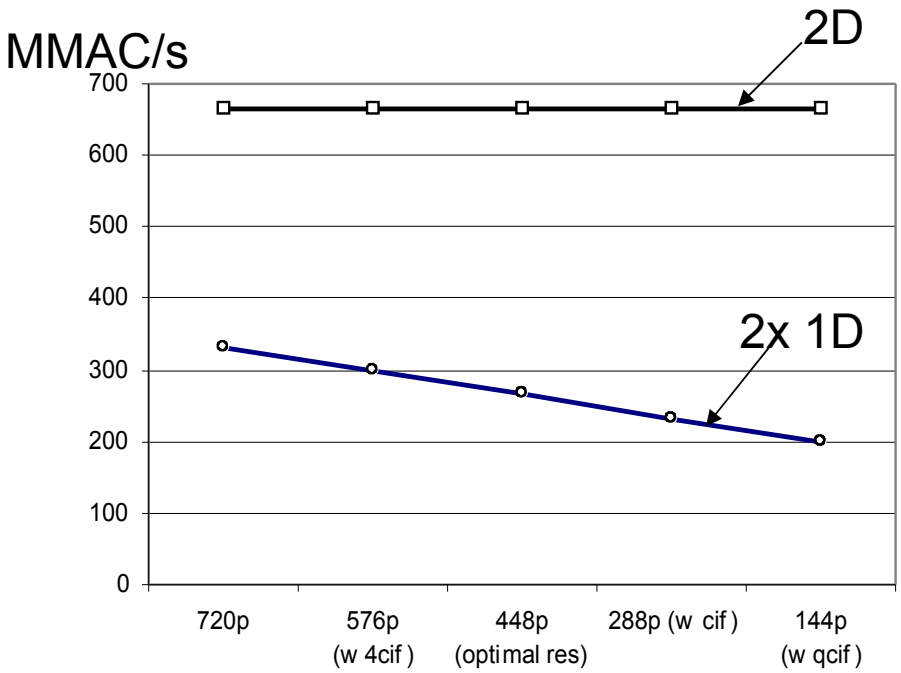





Image resizing





- **Downsampling 720p YUV 4:2:0 at 30 fps:**



Multi-format support

- Codec needs to resize images because of i) screen lay-outs ii) encoding format might be different from 720p.

- Vsport link from camera to codec is 60MB/s. 720p takes 41.5MB/s, so there is some spare b/w.

- Offload codec by having camera providing the same picture in more than one size:
  - one (large) for selfview
  - one picture to be encoded (if different from selfview and b/w allows)
  - one very small image to be used as PIP (if b/w allows)

Format combinations for 16:9

720p

+

w4cif

+

448p

+

wcif

+

+

Format combinations for 4:3

- 4CIF: 4CIF + QCIF
- 448P: 4CIF + 448P + QCIF
- CIF: 4CIF + CIF + QCIF

- 4SIF: 4SIF + QSIF
- 400P: 4SIF + 400P + QSIF
- SIF: 4SIF + SIF + QSIF

Noise reduction

Trade-offs for a noise-reduction filter

- Remove noise
- Avoid too much blurring of edges and details
- Avoid the temporal filtering ”ghost effect”
- Fairly simple because of real-time constraints

Noise reduction

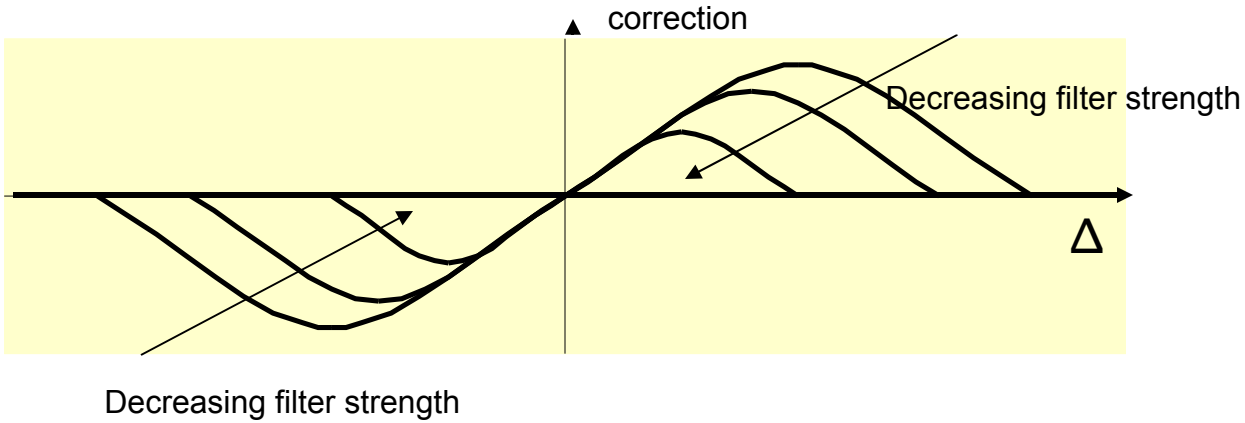
- Combined temporal/spatial filter. Idea (simplified):

  - Let *n*' be a low-pass filtered estimate of pixel *n*, and $\Delta = |n'-n|$

  - *n'* is computed using *n* and its neighbors in time and space.

  - Motion adaptive = The time/space mix depends on motion

  - Adjust n by correction depending on Δ:



Programming model

- FPGA – VHDL
- FPGA softcore - C
- DSP - C

C:

- Code written for readability/maintainability
- Code written for max use of H/W (optimization)

Why DSP code optimization

- 1280 x 720 pixels/frame x 30 frames/s = 27,648,000 pixels/s
- 600 MHz DM642 DSP

Why DSP code optimization

- 1280 x 720 pixels/frame x 30 frames/s = 27,648,000 pixels/s
- 600 MHz DM642 DSP

Bad news:

- 600,000,000 / 27,648,000 = 22 cycles/pixel

Why DSP code optimization

- 1280 x 720 pixels/frame x 30 frames/s = 27,648,000 pixels/s
- 600 MHz DM642 DSP

Bad news:
- 600,000,000 / 27,648,000 = 22 cycles/pixel

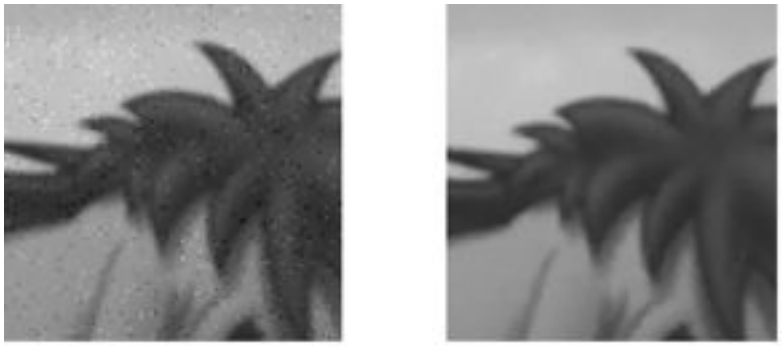
Good news:
- 8 instructions may be issued in parallel
- each containing some ILP

Yet, un-optimized code will easily use 100%+ CPU

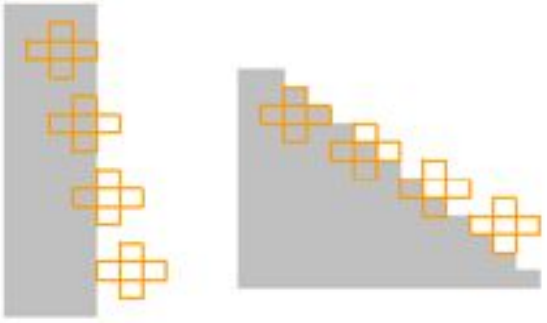
Example: Median filter optimization

- Replace a pixel by the median of the pixel itself and some neighbor pixels. A common non-linear filter for noise removal.

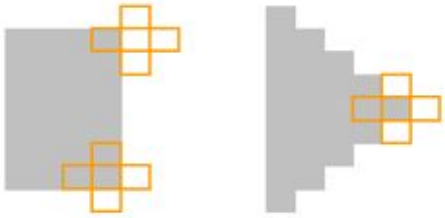
Removes noise:



Preserves edges:

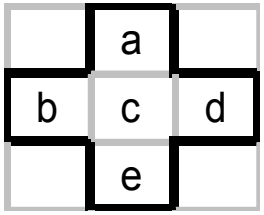


Preserves some (but not all) corners:



Example: Median filter optimization

- A naïve first attempt:




```
int u8cmp(const void *ptr1, const void *ptr2)
{
    return *(unsigned char*)ptr1 - *(unsigned char*)ptr2;
}
...

unsigned char lst[5];
lst[0] = a; lst[1] = b; lst[2] = c; lst[3] = d; lst[4] = e;
qsort(lst, 5, sizeof(unsigned char), u8cmp);
return(lst[2]);
```

Far more than 100% CPU load at 720p30

Example: Median filter optimization

- Find a better computation scheme:

  MEDIAN(a, b, c, d, e) =
  MAX(MIN(MAX(t, c), MAX(MIN(a, b), MIN(d, e))), MIN(t, c)),
  where *t = MIN(MAX(a, b), MAX(d, e))*

- Use fewer function calls
- Work on several pixels at a time to leverage SIMD
- Assert trip count to compiler and hint about unroll
- Pointer alias disambiguation
- ISA targeted optimizations

Using C with intrinsics

```
void median5(
              unsigned char *p0,
              unsigned char *p1,
              unsigned char *p2,
              unsigned char *restrict dst,
              unsigned n
            )
{
    double    a,  b,  c,  d,  e;
    unsigned  a0, b0, c0, d0, e0, q0, r0, s0, t0, u0, v0, x0, y0, z0;
             a1, b1, c1, d1, e1, q1, r1, s1, t1, u1, v1, x1, y1, z1, m0, m1, i;

    #pragma UNROLL(2)
    #pragma MUST_ITERATE(_MEDIAN5_MIN_N/8, , _MEDIAN5_TRP_FACTOR)
    for(i = 0; i < n; i += 8)
    {
        /* ============ Get new pixels from source array ===================== */
                             a = _memd8(&p0[ 0]);                       p0 += 8;
        d = _memd8(&p1[-1]); b = _memd8(&p1[ 0]); e = _memd8(&p1[ 1]); p1 += 8;
                             c = _memd8(&p2[ 0]);                       p2 += 8;
        a0 = _lo(a); b0 = _lo(b); c0 = _lo(c); d0 = _lo(d); e0 = _lo(e);
        a1 = _hi(a); b1 = _hi(b); c1 = _hi(c); d1 = _hi(d); e1 = _hi(e);

        /* ============ Compute eight median values  ======================== */
        q0 = _maxu4(a0, b0);    u0 = a0 + b0 - q0;      r0 = _maxu4(d0, e0);
        v0 =  d0 + e0 - r0;     s0 = _maxu4(u0, v0);    x0 = _minu4(q0, r0);
        t0 = _maxu4(x0, c0);    y0 = x0 + c0 - t0;      z0 = _minu4(_mvd(t0), s0);
        m0 = _maxu4(z0, y0);

        q1 = _maxu4(a1, b1);    u1 =  a1 + b1 - q1;     r1 = _maxu4(d1, e1);
        v1 =  d1 + e1 - r1;     s1 = _maxu4(u1, v1);    x1 = _minu4(q1, r1);
        t1 = _maxu4(x1, c1);    y1 =  x1 + c1 - t1;     z1 = _minu4(_mvd(t1), s1);
        m1 = _maxu4(z1, y1);

        _memd8(&dst[0]) = _itod(m1, m0); dst += 8;
    }
}
```

Compiler feedback:

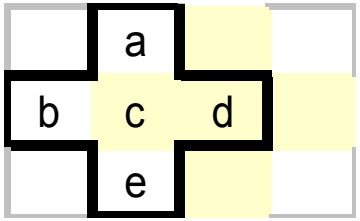
```
;*----------------------------------------------------------------------------*
;*   SOFTWARE PIPELINE INFORMATION
;*
;*      Loop source line                 : 152
;*      Loop opening brace source line   : 153
;*      Loop closing brace source line   : 174
;*      Loop Unroll Multiple             : 2x
;*      Known Minimum Trip Count         : 4
;*      Known Max Trip Count Factor      : 1
;*      Loop Carried Dependency Bound(^) : 0
;*      Unpartitioned Resource Bound     : 13
;*      Partitioned Resource Bound(*)    : 13
;*      Resource Partition:
;*                                A-side   B-side
;*      .L units                    13*      13*
;*      .S units                     1        2
;*      .D units                     6        6
;*      .M units                     3        3
;*      .X cross paths               1        0
;*      .T address paths            12       12
;*      Long read paths              0        0
;*      Long write paths             0        0
;*      Logical  ops (.LS)           0        0     (.L or .S unit)
;*      Addition ops (.LSD)         17       16     (.L or .S or .D unit)
;*      Bound(.L .S .LS)             7        8
;*      Bound(.L .S .D .LS .LSD)    13*      13*
;*
;*      Searching for software pipeline schedule at ...
;*         ii = 13 Schedule found with 3 iterations in parallel
;*      Done
;*
;*      Epilog not entirely removed
;*      Collapsed epilog stages     : 1
;*      Collapsed prolog stages     : 2
```

Assembler:

```
$C$L2:    ; PIPED LOOP KERNEL
$C$DW$L$_median5$3$B:

   [ A2]   SUB     .S1     A2,1,A2           ; <0,24>
|| [ A0]   SUB     .D1     A0,1,A0           ; |152| <0,24>
||         MAXU4   .L1     A4,A3,A4          ; |173| <0,24>
||         MINU4   .L2     B4,B6,B4          ; |173| <0,24>
|| [!A2]   LDNDW   .D2T2   *-B25(8),B19:B18  ; |173| <1,11>
||         ADD     .S2     B16,B22,B26       ; |173| <1,11>

   [!A1]   STNDW   .D2T1   A5:A4,*B21++(16)  ; |173| <0,25>
||         MAXU4   .L2     B4,B8,B4          ; |173| <0,25>
||         MAXU4   .L1     A19,A21,A3        ; |173| <1,12>
||         CMPGTU4 .S2     B22,B16,B6        ; |173| <1,12>
||         AND     .S1     A3,A18,A19        ; |173| <1,12>
||         XOR     .D1X    B20,A3,A18        ; |173| <1,12>

           ADD     .S2     B17,B23,B28       ; |173| <1,13>
||         AND     .S1     A20,A18,A17       ; |173| <1,13>
||         MINU4   .L1     A3,A17,A18        ; |173| <1,13>
||         ADD     .D2     B18,B28,B6        ; |173| <1,13>
||         MAXU4   .L2     B28,B18,B7        ; |173| <1,13>
||         XPND4   .M2     B6,B9             ; |173| <1,13>
||         LDNDW   .D1T1   *+A24(1),A21:A20  ; |173| <2,0>

           ADD     .S2     B19,B29,B27       ; |173| <1,14>
||         MAXU4   .L2     B29,B19,B29       ; |173| <1,14>
||         MAXU4   .L1     A18,A5,A17        ; |173| <1,14>
||         SUB     .D2     B6,B7,B6          ; |173| <1,14>
||         OR      .S1     A17,A19,A19       ; |173| <1,14>
||         LDNDW   .D1T1   *-A24(1),A5:A4    ; |173| <2,1>

           XOR     .S2     B20,B9,B17        ; |173| <1,15>
||         MAXU4   .L2     B23,B17,B8        ; |173| <1,15>
||         SUB     .S1     A16,A3,A7         ; |173| <1,15>
||         MVD     .M1     A17,A9            ; |173| <1,15>
||         SUB     .D1     A8,A19,A3         ; |173| <1,15>
||         MINU4   .L1     A19,A7,A8         ; |173| <1,15>
||         LDNDW   .D2T1   *B24++(16),A19:A18 ; |173| <2,2>

           XOR     .S2     B20,B9,B17        ; |173| <1,15>
||         MAXU4   .L2     B23,B17,B8        ; |173| <1,15>
||         SUB     .S1     A16,A3,A7         ; |173| <1,15>
||         MVD     .M1     A17,A9            ; |173| <1,15>
||         AND     .S2     B9,B22,B16        ; |173| <1,16>
||         AND     .D2     B16,B17,B17       ; |173| <1,16>
||         MINU4   .L2     B8,B29,B9         ; |173| <1,16>
||         ADD     .S1     A8,A4,A4          ; |173| <1,16>
||         MAXU4   .L1     A8,A4,A8          ; |173| <1,16>
||         LDNDW   .D1T1   *A24++(16),A21:A20 ; |173| <2,3>

           SUB     .S2     B28,B8,B17        ; |173| <1,17>
||         MAXU4   .L2     B9,B19,B16        ; |173| <1,17>
||         OR      .D2     B17,B16,B8        ; |173| <1,17>
||         SUB     .S1     A4,A8,A3          ; |173| <1,17>
||         MAXU4   .L1     A3,A6,A6          ; |173| <1,17>
||         MVD     .M1     A8,A22            ; |173| <1,17>
||         LDNDW   .D1T2   *-A24(7),B19:B18  ; |173| <2,4>

   [ A0]   B       .S2     $C$L2             ; |152| <0,31>
||         MVD     .M2     B16,B17           ; |173| <1,18>
||         ADD     .S1     A18,A5,A7         ; |173| <1,18>
||         MAXU4   .L1     A7,A9,A8          ; |173| <1,18>
||         SUB     .D2     B26,B8,B8         ; |173| <1,18>
||         MINU4   .L2     B8,B7,B7          ; |173| <1,18>
||         LDNDW   .D1T2   *-A24(8),B17:B16  ; |173| <2,5>

           SUB     .D1     A7,A17,A23        ; |173| <1,19>
||         ADD     .S2     B7,B18,B18        ; |173| <1,19>
||         MAXU4   .L2     B7,B18,B7         ; |173| <1,19>
||         ADD     .S1     A20,A4,A4         ; |173| <2,6>
||         MAXU4   .L1     A4,A20,A7         ; |173| <2,6>
||         LDNDW   .D2T2   *-B24(8),B23:B22  ; |173| <2,6>

           SUB     .S2     B27,B29,B7        ; |173| <1,20>
||         SUB     .D2     B18,B7,B8         ; |173| <1,20>
||         MAXU4   .L2     B8,B6,B6          ; |173| <1,20>
||         MVD     .M2     B7,B4             ; |173| <1,20>
||         LDNDW   .D1T2   *-A24(9),B29:B28  ; |173| <2,7>
||         MAXU4   .L1     A5,A21,A17        ; |173| <2,7>
||         ADD     .S1     A21,A5,A5         ; |173| <2,7>

           ADD     .S2     B9,B19,B7         ; |173| <1,21>
||         MAXU4   .L2     B17,B7,B9         ; |173| <1,21>
||         MINU4   .L1     A9,A8,A9          ; |173| <1,21>
||         LDNDW   .D2T1   *B25++(16),A5:A4  ; |173| <2,8>
||         ADD     .S1     A20,A18,A8        ; |173| <2,8>
||         ADD     .D1     A21,A19,A16       ; |173| <2,8>

           SUB     .S2     B7,B16,B7         ; |173| <1,22>
||         MINU4   .L2     B17,B9,B9         ; |173| <1,22>
||         MAXU4   .L1     A9,A23,A5         ; |173| <1,22>
||         SUB     .D1     A5,A17,A9         ; |173| <2,9>
||         CMPGTU4 .S1     A18,A20,A23       ; |173| <2,9>
```

Median filter optimization

- Reuse MIN/MAX comparison of c & d in next MEDIAN computation?



(same symmetry also in the vertical direction)

- Unroll vertically to save some memory reads?
- Try to be efficient with regards to cache, memory bank conflicts etc.
- Merge with sourounding computations?
- Other ideas?

… or move computation to FPGA

```
  signal min1       : unsigned(PIX_W-1 downto 0) := (others => '0');
  signal min3       : unsigned(PIX_W-1 downto 0) := (others => '0');
  signal min4       : unsigned(PIX_W-1 downto 0) := (others => '0');
  signal min4_r1    : unsigned(PIX_W-1 downto 0) := (others => '0');
  signal min5       : unsigned(PIX_W-1 downto 0) := (others => '0');

begin  -- architecture rtl

  ---------------------------------------------------------------------------
  --* Stage 1: Calculate max0 (=max(a,b), min0 (=min(a,b)), max1 (=max(d,e))
  --* and min1 (=min(d,e).
  ---------------------------------------------------------------------------
  stage1_pr : process(clk)
  begin
    if (rising_edge(clk)) then
      if (ce = '1') then
        actdata_r1 <= actdata;
        if (enable = '1') then
          -- max0 and min0
          if (a > b) then
            max0 <= unsigned(a);
            min0 <= unsigned(b);
          else
            max0 <= unsigned(b);
            min0 <= unsigned(a);
          end if;
          -- max1 and min1
          if (d > e) then
            max1 <= unsigned(d);
            min1 <= unsigned(e);
          else
            max1 <= unsigned(e);
            min1 <= unsigned(d);
          end if;
          -- Pipeline delay
          c_r1 <= unsigned(c);
        else                            -- Delay input when bypassing
          max0 <= unsigned(b);
        end if;
      end if;
    end if;
  end process stage1_pr;

  ---------------------------------------------------------------------------
  --* Stage 2: Calculate max2 (=max(min0,min1)) and min3 (=min(max0,max1))
  ---------------------------------------------------------------------------
  stage2_pr : process(clk)
  begin
    if (rising_edge(clk)) then
      if (ce = '1') then
        actdata_r2 <= actdata_r1;
```

TANDBERG Precision HD Camera

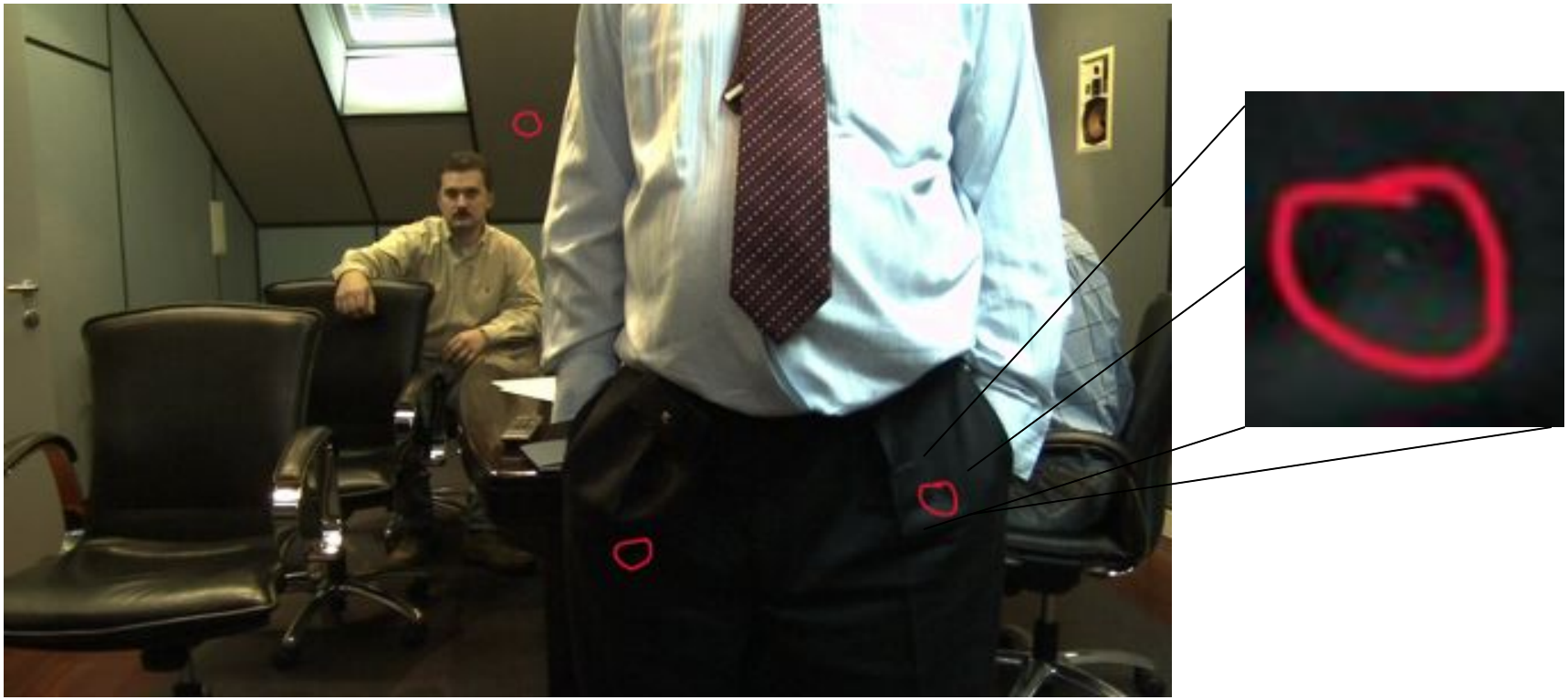
”the Tiger Camera”

S/W upgrade, example

- Some bad pixels in every sensor
- Most annoying are so-called hot pixels
- Thought we could manage without s/w correction

- But decided to make s/w upgrade
  - Feedback from customers
  - Manufacturing yield

Hot pixels

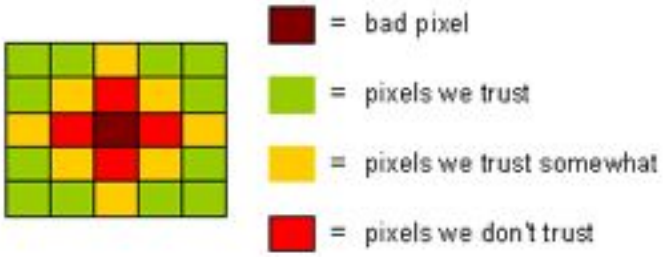


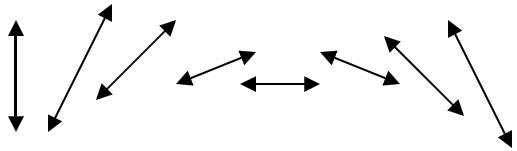
Hot pixels

- **Challanges:**
  - Blind correction not good enough
  - Need to calibrate & store list of bad addresses
  - Need to correct as early as possible in the chain
  - But no room in FPGA
  - When image arrives at DSP it has already gone through demosaicking and 4:4:4 to 4:2:2 conversion
  - A hot pixel has diffused out to its neighbors
- **Solution:**
  - Correct limited # of pixels (50) in DSP
  - Y only and hope median/noise-filter will take the rest
  - Correct some neighbors too
  - Be careful and don't try to correct 100%

Interpolating hot pixels

- We use brute force, checking variation in 8 different directions using differences of trusted pixels





- If surrounding pixels vary a lot, don't interpolate as the interpolated value might be a poor estimate of the true value
- Smooth transition between interpolation and no interpolation:

S/W upgrades

- Every codec s/w upgrade has contained new camera s/w
- 5 – 10 software upgrades since launch
- Tunings, differences in sensor h/w, codec offload

Product Development in Tandberg

Product Development in Tandberg

- We spend very little energy on things that are not essential
- We do not follow a plan, we do not need procedures
- We do not write hours, we do not measure project cost
- Decisions are delayed as much as possible
- To fail is OK, therefore we deliver spectacular stuff
- Doers are very much respected in Tandberg
- Our goal is to only hire exceptional people
- Communication is a key skill for all our engineers
- We are fast and “sloppy”
- Slack is embedded, and “skunk work” projects appreciated
- Fun and Profit
- Agile and Lean

Product Development in Tandberg

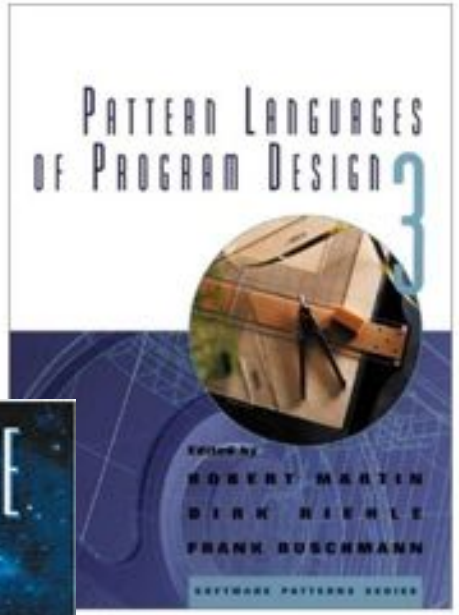
- We spend very little energy on things that are not essential
- We do not follow a plan, we do not need procedures
- We do not write hours, we do not measure project cost
- Decisions are delayed as much as possible
- To fail is OK, therefore we deliver spectacular stuff
- Doers are very much respected in Tandberg
- Our goal is to only hire exceptional people
- Communication is a key skill for all our engineers
- We are fast and “sloppy”
- Slack is embedded, and “skunk work” projects appreciated
- Fun and Profit
- Agile and Lean

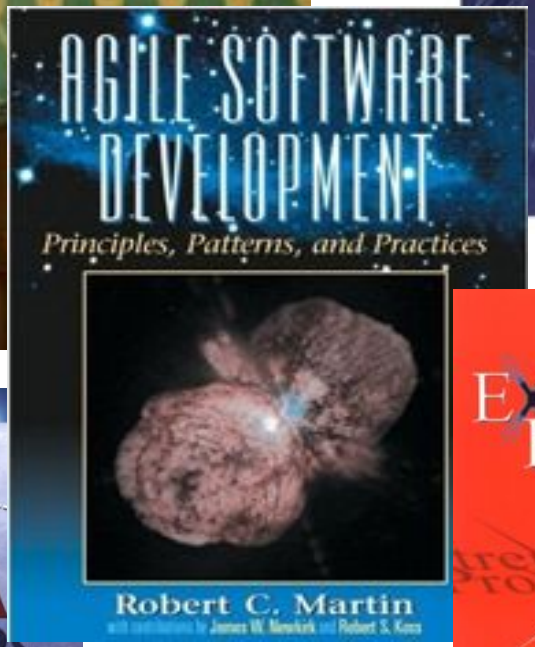
We follow principles, not processes!

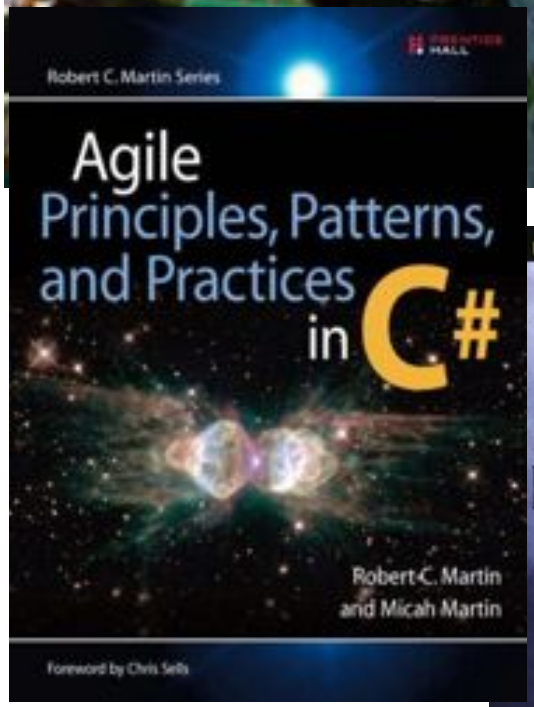

QCon
InfoQ
JAOO

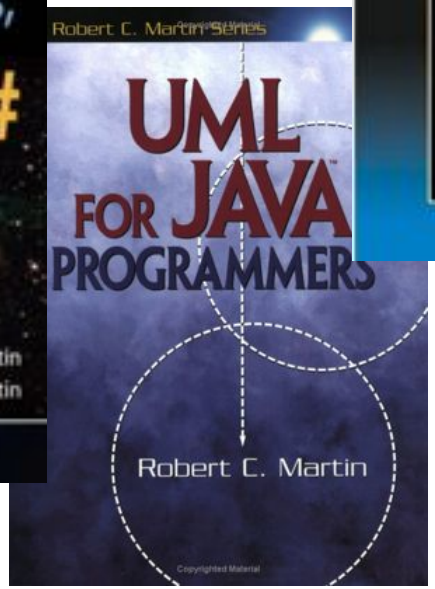

SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
A Pattern Language for Distributed Computing
Volume 4
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

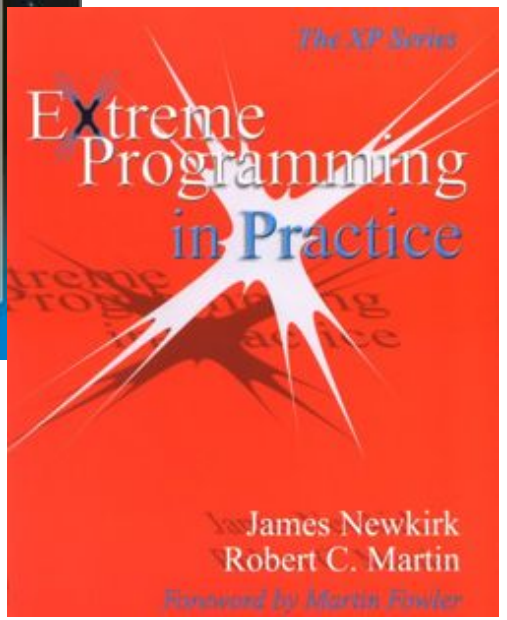

SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED SOFTWARE ARCHITECTURE
On Patterns and Pattern Languages
Volume 5
Frank Buschmann
Kevlin Henney
Douglas C. Schmidt


Pattern Languages of Program Design 3
Robert Martin
Dirk Riehle
Frank Buschmann

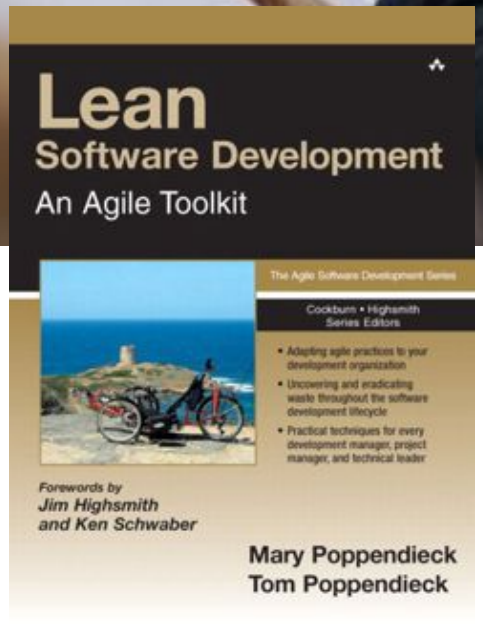

Agile Software Development
Principles, Patterns, and Practices
Robert C. Martin


Robert C. Martin Series
Agile Principles, Patterns, and Practices in C#
Robert C. Martin and Micah Martin
Foreword by Chris Sells


Robert C. Martin Series
UML for Java Programmers
Robert C. Martin


The XP Series
Extreme Programming in Practice
James Newkirk
Robert C. Martin
Foreword by Martin Fowler


XP2006


Lean
Software Development
An Agile Toolkit
The Agile Software Development Series
Cockburn • Highsmith
Series Editors
• Adapting agile practices to your development organization
• Uncovering and eradicating waste throughout the software development lifecycle
• Practical techniques for every development manager, project manager, and technical leader
Forewords by
Jim Highsmith
and Ken Schwaber
Mary Poppendieck
Tom Poppendieck


The Addison Wesley Signature Series
Implementing
Lean Software
Development
From Concept to Cash
Mary and Tom
Poppendieck

“I have been working with software development groups all around the world, and you are way ahead of most. Actually this is one of the few companies that I would consider working for.”

(a consultant visiting our R&D department)

Questions ?

www.tandberg.com

mattias.ahnoff@tandberg.com
olve.maudal@tandberg.com