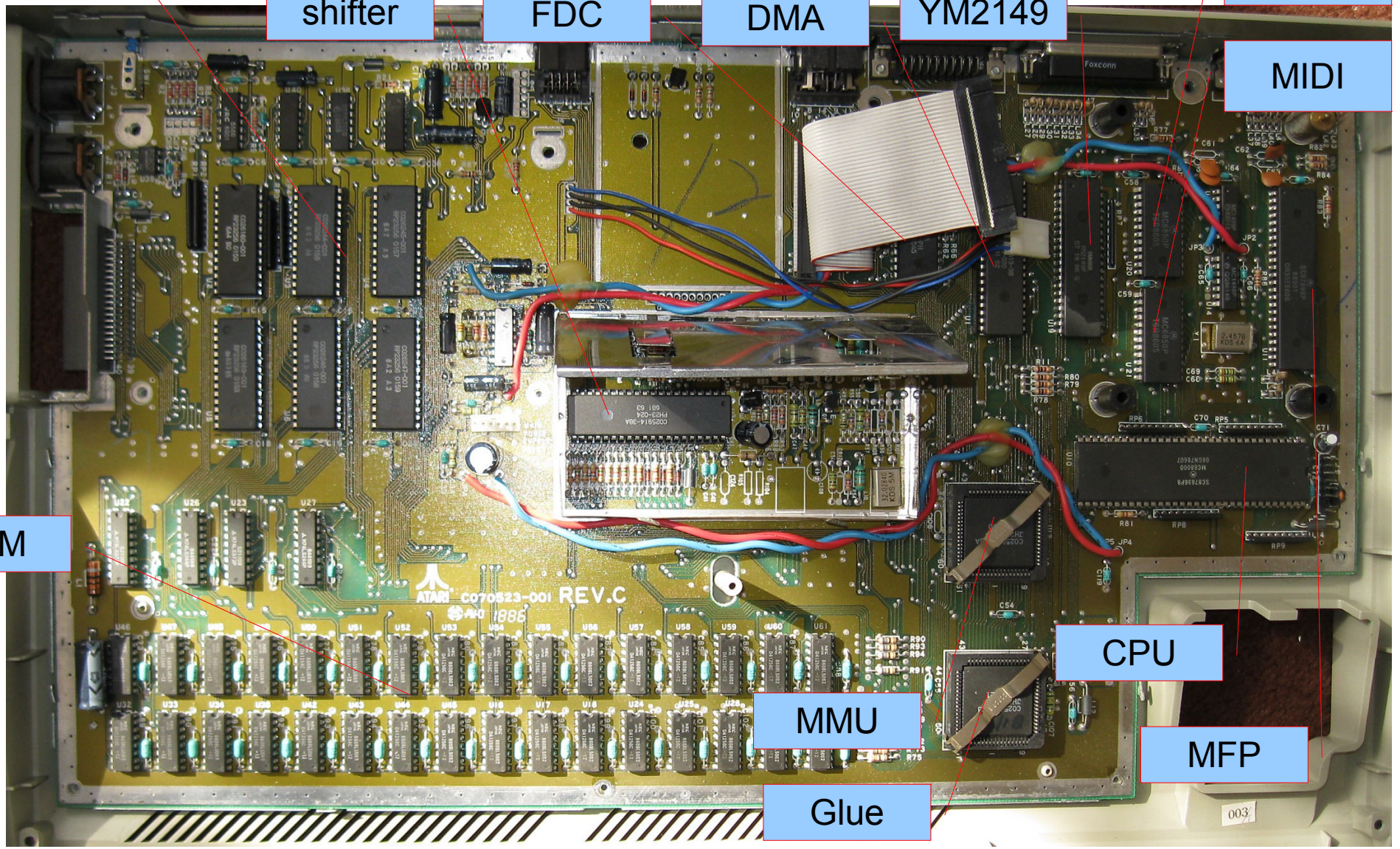
Atari ST knowhow

Note that the original presentation uses special effects which are not rendered in this PDF.

Workshop 2:
Hardware and software hacks

hackerspace.lu

What Hardware?



Add RAM, RTC, HD, VIDEO...

Software hacks?

Need to know:

- CPU
- Memory layout
- Video modes
- Peripheral registers

The CPU

- MC68000 @ 8Mhz
- Orthogonal?

  56 instructions / 14 address modes / 16 registers

  almost combinable each with each one
- Attention: big-endian (i.e. **not** Intel)
- 32 bit (but 16 bit bus)
- User mode / system mode

68000 assembly

Instructions

- Bcc, BRA, BSR, RTS, JMP, DBcc
- MOVE, MOVEQ MOVEM, MOVEP
- ADD, SUB, MUL, DIV
- AND, OR, EOR, NOT
- ASL, LSL, ROL, ROXL
- BTST, BSET, BCLR, BCHG
- SWAP, *BCD
- * SR, TRAP, TAS
- And many more....

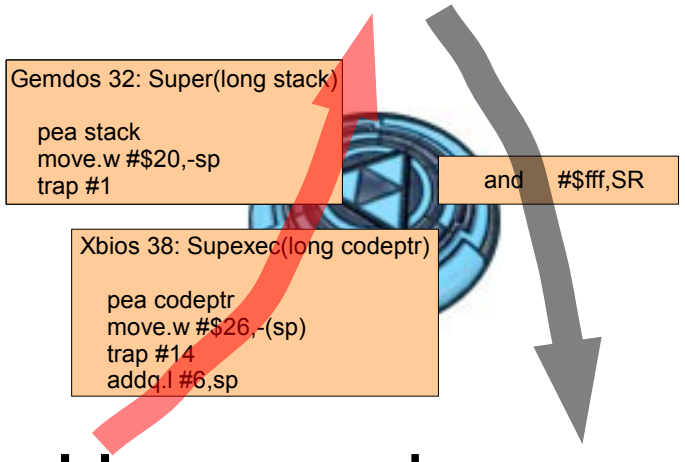
Address modes

- Dn
- An
- (An)
- (An)+
- -(An)
- x(An)
- x(An,Dn)
- xxx.W
- xxx.L
- x(PC)
- x(PC,Dn)
- #xxx
- SR

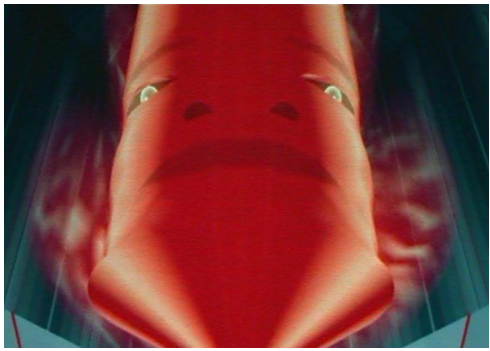
Memory layout & privileges

```
00 0000   | ROM        |  Reset:  Supervisor Stack Pointer
00 0004   | ROM        |  Reset:  Program Counter
00 0008   | RAM        |  0 Kbyte RAM
00 0800   | RAM        |  2 Kbyte RAM          (start of user ram)
08 0000   | RAM        |  512 Kbyte RAM

10 0000   | RAM        |  1 Mbyte RAM

          ---------------
fa 0000   | ROM        |  320 Kbyte ROM
          ---------------
          ---------------
fc 0000   | ROM        |  Reset:  Supervisor Stack Pointer
fc 0004   | ROM        |  Reset:  Program Counter
fc 0008   | ROM        |  192 Kbyte ROM
fe ffff   | ROM        |  0 Kbyte ROM
          ---------------
          ---------------
ff 8000   | I/O        |  Configuration Registers
ff 8200   | I/O        |  Display Registers
ff 8400   | I/O        |  Reserved
ff 8600   | I/O        |  DMA/Disk Registers
ff 8800   | I/O        |  Sound Registers
          ---------------
          ---------------
ff fa00   | I/O        |  MC68xxx Registers
ff fc00   | I/O        |  MC68xx Registers
          ---------------
```

Supervisor mode



User mode



http://www.atari-forum.com/viewtopic.php?f=68&t=10137#p79672

Video chip

- Nice mix between shifter MMU and Glue
  - Restriction: the screen base address has the low byte always $00, i.e. only 256 byte frontier
  - Restriction: changing base address only has effect on next frame
- No direct colors on screen, but indexes into 16 colors palette
- Changing color value has immediate effect
- Think GIF but 16 instead of 256 colors

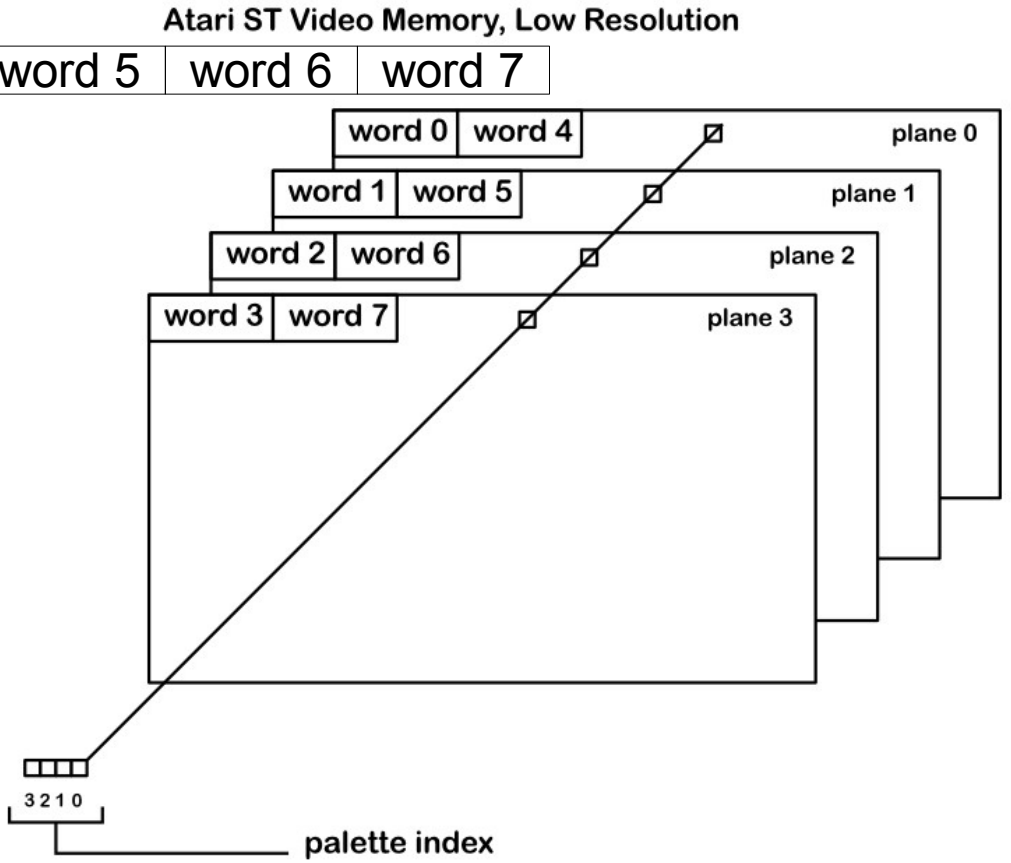
VIDEO memory

Word interleaved with color lookup table

word 0	word 1	word 2	word 3	word 4	word 5	word 6	word 7

Memory layout

320x200x4=256000bit=32000 bytes



Grafic bitplanes

1st pixel is composed of

high order bits from words 0-3

(that's **not** GIF anymore)

Colorcycling / screen changing

- Uses no CPU
- Uses no additional RAM
- Very basic step animation up to 16 frames

  => neochrome pictures

- 512K RAM allows

  16 different pictures
- 1MB has 32 steps
- Complex animations

=> shiny bubbles demo

Both can be combined

=> pandemonium acid screen

Moving grafics

- up / down : welcome to movem.l instruction!
- 32000/200 = 160 bytes per line

```
move.l $44E,a0              ; _v_bas_ad
```

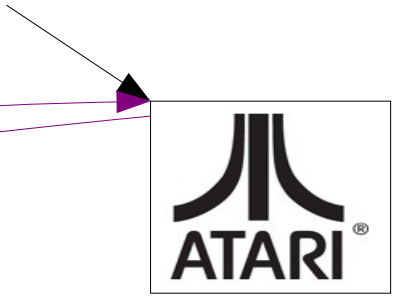

```
movem.l 160(a0),d0-d7       ; 32 bytes
movem.l d0-d7,(a0)+         ; repeat 5 times
```

Address mode not allowed, oh oh....

Moving grafics

- up / down : welcome to movem.l instruction!
- 32000/200 = 160 bytes per line

```
move.l $44E,a0              ; _v_bas_ad
lea 160(a0),a0


movem.l (a0)+,d0-d7         ; 32 bytes
movem.l d0-d7,-160-32(a0)   ; repeat 5 times
```



Moving grafics

- up / down : welcome to movem.l instruction!
- 32000/200 = 160 bytes per line

```
    move.l $44E,a0              ; _v_bas_ad
    lea 160(a0),a0
    moveq #4,d0       ; repeat 5 times
cpline:
    movem.l (a0)+,d1-d7/a1   ; 32 bytes
    movem.l d1-d7/a1,-160-32(a0)
    dbf d0,cpline
```

Moving grafics

- We copied 1 line, now copy whole screen

```
    move.l $44E,a0              ; _v_bas_ad
    lea 160(a0),a0
    move.w #199,d1
cpscreen:
    ; screen wraps around at end of line
    moveq #4,d0             ; repeat 5 times
cpline:
    movem.l (a0)+,d2-d7/a1-a2  ; 32 bytes
    movem.l d2-d7/a1-a2,-160-32(a0)
    dbf d0,cpline
    dbf d1,cpscreen
```

Moving grafics

- This can be optimized: 5*200=1000

```
    move.l $44E,a0              ; _v_bas_ad
    lea 160(a0),a0
    move.w #999,d0              ; repeat 1000x
cpscreen:
    movem.l (a0)+,d2-d7/a1-a2   ; 32 bytes
    movem.l d2-d7/a1-a2,-160-32(a0)
    dbf d0,cpscreen
```

scrolling

- Bring in new data on bottom

```
   move.l $44E,a0          ; _v_bas_ad
   lea tmpbuf,a1     ; save top line
   moveq #4,d0             ; repeat 5 times
bckppline:
   movem.l (a0)+,d1-d7/a2  ; 32 bytes
   movem.l d1-d7/a2,(a1)
   lea 32(a1),a1
   dbf d0,bckppline
   move.l $44E,a0          ; _v_bas_ad
   lea 160(a0),a0
   move.w #999-5,d0        ; repeat 1000x -1 line
cpscreen:
   movem.l (a0)+,d2-d7/a1-a2  ; 32 bytes
   movem.l d2-d7/a1-a2,-160-32(a0)
   dbf d0,cpscreen
```

```
   move.l $44E,a0          ; _v_bas_ad
   adda.l #32000-160,a0
   lea tmpbuf,a1
   moveq #4,d0             ; repeat 5 times
cplineback:
   movem.l (a1)+,d1-d7/a2  ; 32 bytes
   movem.l d1-d7/a2,(a0)
   lea 32(a0),a0
   dbf d0,cplineback
   [...]
   BSS
tmpbuf:
   ds.b 160
```

scrolling

- We can make that shorter: sub routine

```
    move.l $44E,a0          ; _v_bas_ad
    lea tmpbuf,a1
    bsr linecopier
    move.l $44E,a0          ; _v_bas_ad
    move.w #199-1,d0          ; repeat 199x
cpscreen:
    bsr linecopier
    dbf d0,cpscreen
    move.l $44E,a0          ; _v_bas_ad
    adda.l #32000-160,a1
    lea tmpbuf,a0
    bsr linecopier
    [...]
Linecopier:             ; copy 32 bytes from (a0) to (a1)
    moveq #4,d0           ; repeat 5 times
cpline:
    movem.l (a0)+,d1-d7/a2  ; 32 bytes
    movem.l d1-d7/a2,(a1)
    lea 32(a1),a1
    dbf d0,cpline
    rts
    BSS
tmpbuf:
    ds.b 160
```

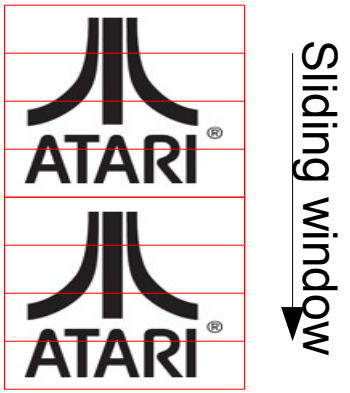
This is slow!

Same scrolling, faster

We could scroll 8 lines at a time

oh, wait... 8*160=1280 and 1280 / 256 = 5

if we move the screen base address...



copy 2 times the picture and

have the screen move around

Apparent movement

Other direction

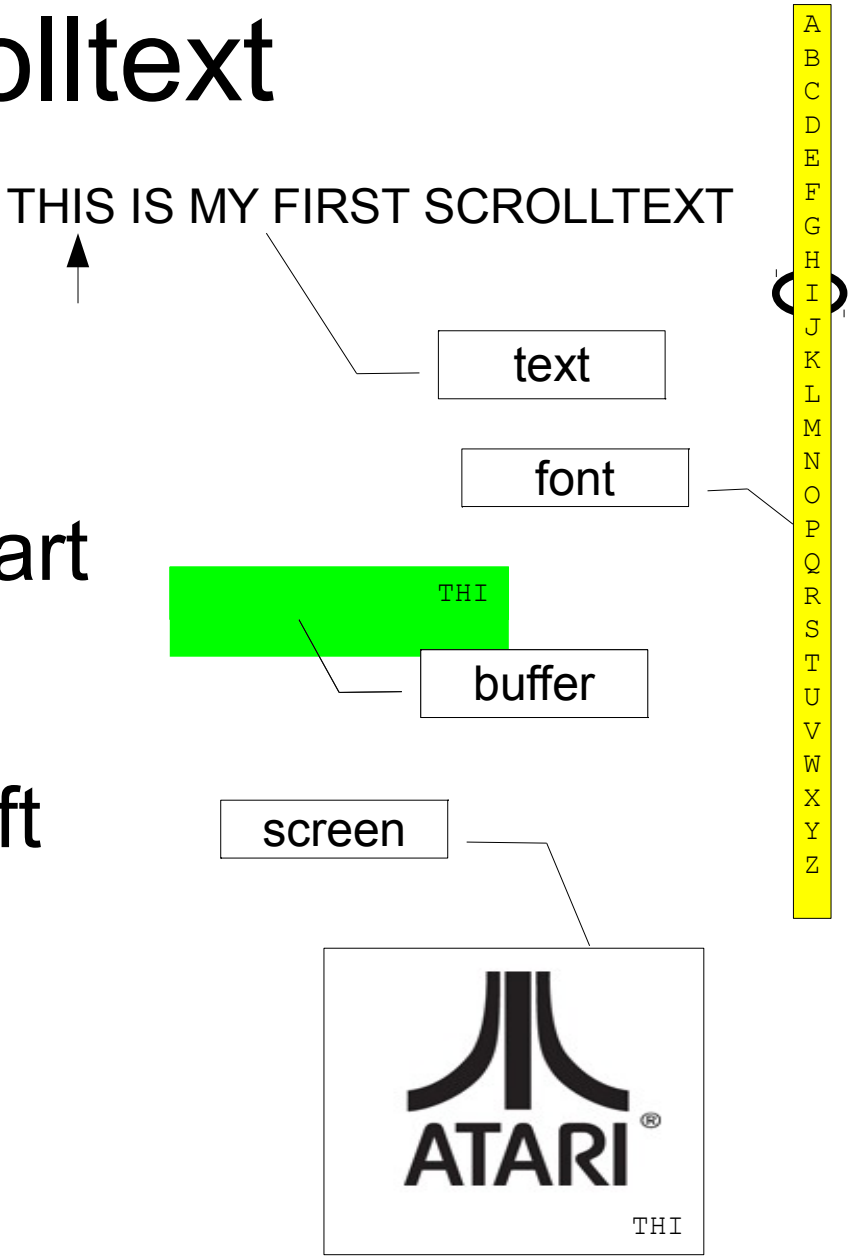
- Right to left, always right to left
- Picture
- Scrolltext
  - Just a very long image
  - Individual letters
- Need to have font handling routine

Horizontal scrolling

- 16 bit boundaries issue
- Scroll speed: the fastest is the easiest
- Speed/Memory tradeoff
  - Shifting data: slow
  - Preshifting: memory

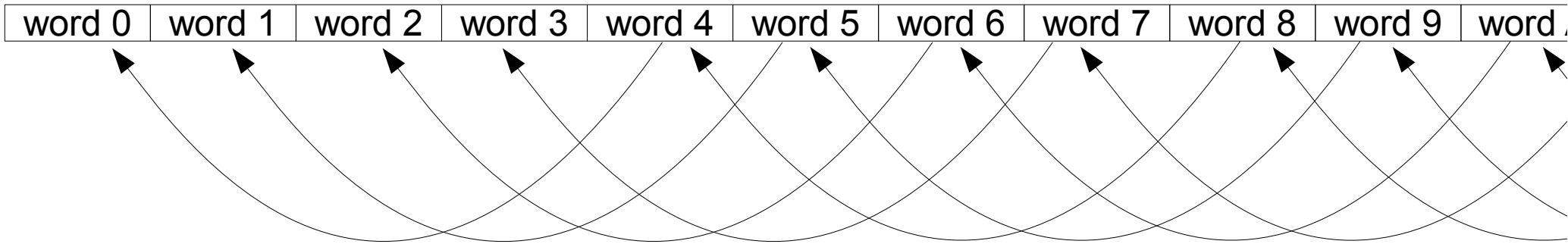
Classic scrolltext

1) Read next text letter
2) Locate font data
3) Copy font data to buffer start
4) Copy buffer to screen
5) Shift buffer one position left
6) Wait vbl



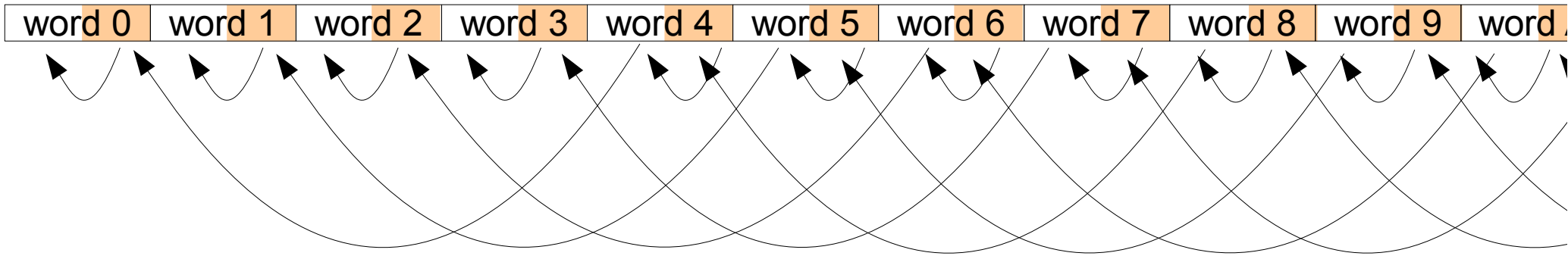
How to shift

- Word shift: move each line 4 words left
  - Every letter is 16 pixels wide
  - That's the fast and easy scroll, it's also fast



How to shift

- Byte shift is slower but easier readable
  - Letters are 8 pixels wide
  - More complex code



How to shift

- Bit shift: very smooth, lotsa CPU
  - Need to lsl all the data
  - Or multiple buffers, extreme case: 16
  - Maybe also have the font data on 16 shifts
  - Speed memory tradeoff: scroll by 2 pixels

Distorter

- A logo moving left and right
- Add a wave = move each line individually
- Quite slow
- Preshifting is the key => 16 times the memory
- Left right boders: what about background?
- Masking with AND & OR

sprites

- A grafical object moving in all directions
- Example: mouse cursor
- Can be hollow, needs complete masking

The bitplane trick

- Less colors = less planes
- Less planes = faster code
- Free planes for other grafics = no masking
- Create cool transparency for free

Colors, more colors

- Pallette modification is immediate
- Favorite register: color 0, also shows in border
- Rasters
- Palette switching in middle of screen

  => 2x16 colors or more... 200x16?
- Spectrum 512: 200 x 3 x 16 (48 / line)

Higher resolution

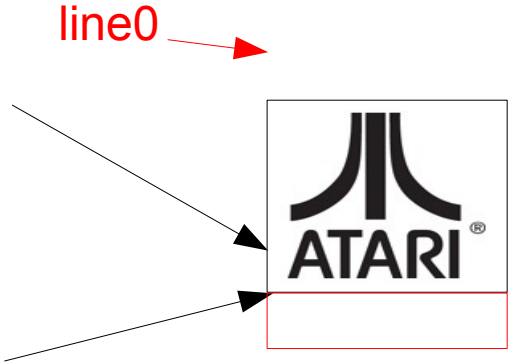
- Ugly border, only usable by background color
- Bottom border, how is it possible?

  V=vertival video (pixels are sent to screen)

  What the GLUE does on bottom screen:
- If line=234 && freq=60 then disable V

Switch here to 60Hz

- If line=259 && freq=50 then disable V



43 lines more

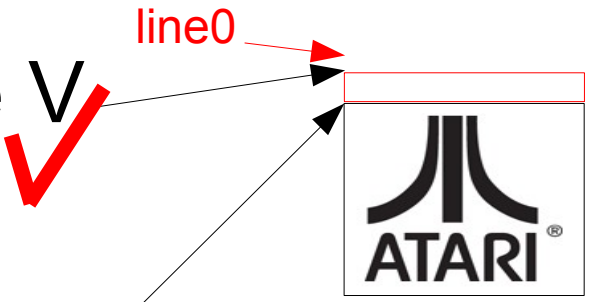
Shiraz didn't expect that...

Top border also

- Same thing, other play
- Same tests are done on top of screen

Switch here to 60Hz

- If line=34 && freq=60 then enable V

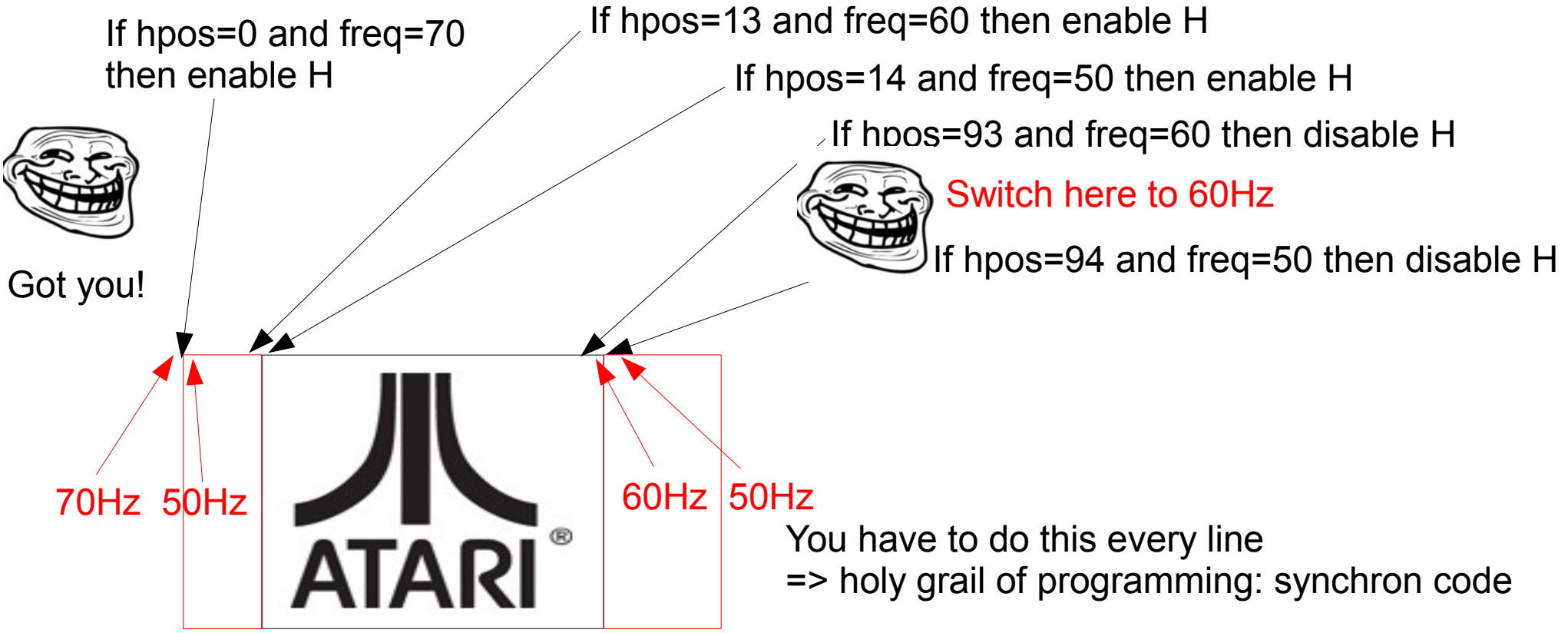


- If line=59 && freq=50 then enable V

25 lines more

And left/right borders

- Simply place some swithces
  - Simply? Some?



You have to do this every line
=> holy grail of programming: synchron code

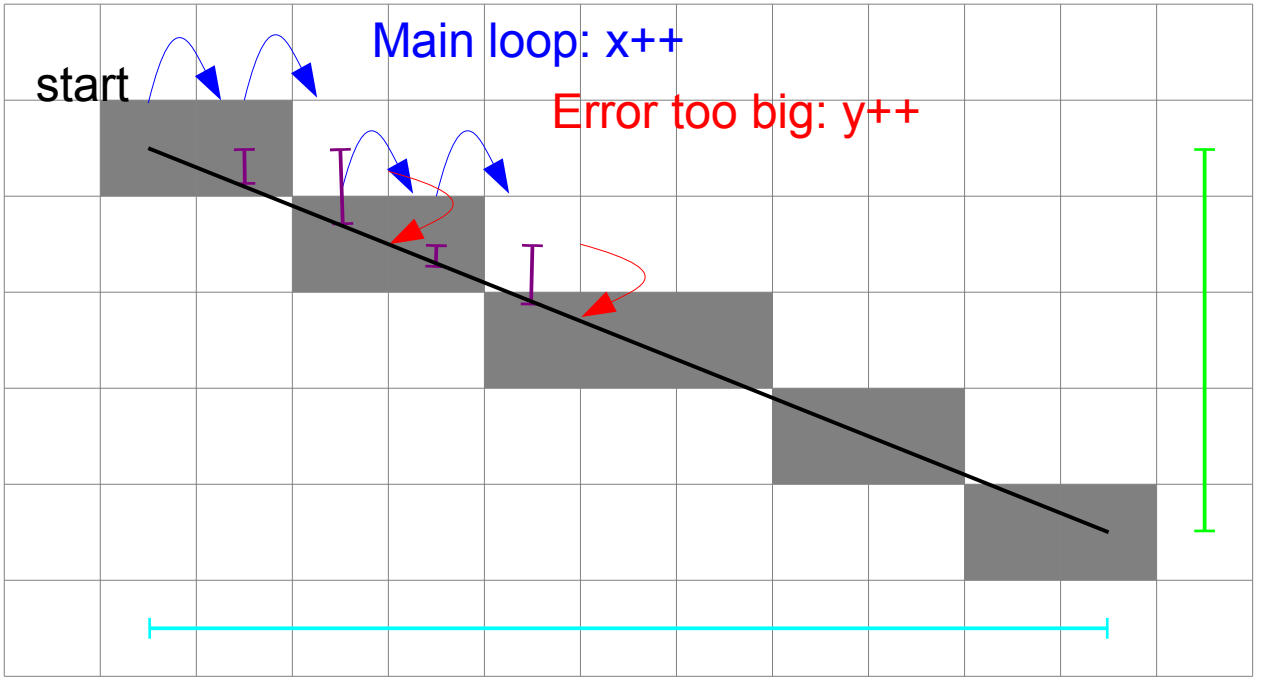
The Sync Scroll

- Invented by... TCB, perfected by Sync :-)
- Hardware scrolling: overcome the 256 limit
- Overscan “eats” data
- More or less lines overscan “move” display

Lines

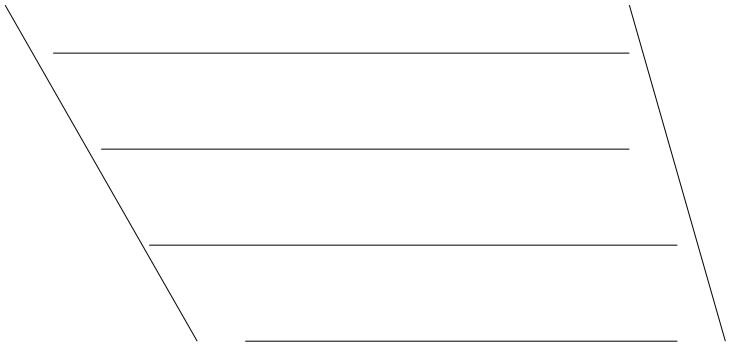
- Yes! no grafic chip, no GL
- How to draw a line?
- Check http://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm

```
deltaerr=ylen/xlen (float)
while x++ < xmax
    plot(x,y)
    error := error + deltaerr
    if error ≥ 0.5 then
        y := y + 1
        error := error - 1.0
```



polygons

- Polygon is 2 lines, filled in between horizontally



- 3D grafics
  - calculate coordinates
  - Check hidden / sort in Z
  - Draw lines or polygons

Digisound

- chiptune part will be handled in the radio show

- Output samples? Use volume registers
- 3 registes of 16 logarythmic values
- Need conversion table
- Slow: need to write 3 times on same address
- Fast: use shadow registers

Multiple voices

- Amiga has 4 voices in hardware
  - Variable speed
  - Volume control
- Brute force calculation: ST plays 4 voices too

  but uses almost 100% CPU
- Comes to the rescue: bresenham routine

  Line with variable slope <=> sample with variable speed
- ST plays 8 voices and still has CPU time left

  see: http://www.atari-forum.com/viewtopic.php?f=68&t=20168

  (or wait for next workshops to digg deeper)

Next Workshops

This is not finalized

3. Putting it all together
4. Assembler and Hardware
5. Algorithms and Optimizations
6. Sync programming
7. First own steps