!252-0286-00L
"The hope is that the progress in hardware...
>...will cure all software ills. However, a [critic] may observe that
>software manages to outgrow hardware in size and sluggishness."
>
- Martin Reiser (1991), "The Oberon System", Addison Wesley
>
>
>paraphrased as
>
>"Software [gets] slower more rapidly than hardware becomes faster"
>
- Niklaus Wirth (Feb. 1995), "A Plea for Lean Software", IEEE Computer

ETH Vorlesung Systembau / Lecture System Construction
>Case Study: Custom designed Single-Processor System
- [First lecture: Custom-designed Single-Processor: the RISC Architecture]
- [Second lecture: Project Oberon on RISC]
- This lecture: Supplemental Material
>
>Review
- detailed study of a GUI workstation system built from the logic gates up
- choices, trade-offs & optimisation with hardware's limited resources
- flexibility in implementing even standard H/W (e.g. SPI)
- transition to self-hosting ("eating one's own dogfood" - Microsoft)
- clean, well-designed extensible API with low surface area
- text, graphical, hardware and network operations with low bureaucracy

FPGA Mouse SW/HW Co-design Example from Project Oberon
- typical PS/2 (or USB) mouse operation involves separation of SW and HW duties
- hardware receives data serially into shift register
- movement packet transmitted byte-by-byte with start, parity and stop bits
- byte reception interrupts processor, which stores bytes in a buffer
- software interprets packets and updates mouse pointer position
- software needs to ensure buffer does not become full (seen in practice)
- is there a better way?

PS/2 Mouse Driver in Hardware
- inspired by Niklaus Wirth's MouseX direct quadrature interface
- reports current mouse position to software
- suggests: receive entire PS/2 packet into a single shift register
- pick out the needed bits, discard others
- can even send F4 initialisation byte
- (by pre-initialising shift register)
- later even enhanced to handle Microsoft wheel button enable

Video (VGA) Interfacing - Worked Example
- how is a video signal produced? VGA (analogue) vs HDMI (digital)
- worked example of an application implemented solely in hardware
- introducing Lola, Niklaus Wirth's HDL (demo)
- 1. generate horizontal & vertical sync and fixed pattern
- 2. create testbed for simulation - VIDExTest.v
- 3. add border and cursor
- 4. input I2S data and store in BRAM
- 5. display BRAM data

End

https://inf.ethz.ch/personal/wirth/ProjectOberon/SourcesVerilog/MouseP.v

```
`timescale 1ns / 1ps  // PS/2 Logitech mouse PDR 14.10.2013 / 8.9.2015
module MouseP(
  input clk, rst,
  inout msclk, msdat,
  output [27:0] out);

  reg [9:0] x, y;
  reg [2:0] btns;
  reg Q0, Q1, run;
  reg [31:0] shreg;
  wire shift, endbit, reply;
  wire [9:0] dx, dy;

// 3 3 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1                 bit
// 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0 9 8 7 6 5 4 3 2 1 0
// ================================================================
// p y y y y y y y y 0 1 p x x x x x x x 0 1 p Y X t s 1 M R L 0 normal
// ----------------------------------------------------------------
// p ----response--- 0 1 --InitBuf echoed--- 1 1 1 1 1 1 1 1 1 1 1 init
// ----------------------------------------------------------------
// p = parity (ignored); X, Y = overflow; s, t = x, y sign bits

  // initially need to send F4 cmd (start reporting); add start and parity bits
  localparam InitBuf = 32'b11111111111111111111110_11110100_0;
  assign msclk = ~rst ? 0 : 1'bz;  // initial drive clock low
  assign msdat = ~run & ~shreg[0] ? 0 : 1'bz;
  assign shift = Q1 & ~Q0;  // falling edge detector
  assign reply = ~run & ~shreg[11];  // start bit of echoed InitBuf, if response
  assign endbit = run & ~shreg[0];  // normal packet received
  assign dx = {{2{shreg[5]}}, shreg[7] ? 8'b0 : shreg[19:12]};  //sign+overfl
  assign dy = {{2{shreg[6]}}, shreg[8] ? 8'b0 : shreg[30:23]};  //sign+overfl
  assign out = {run, btns, 2'b0, y, 2'b0, x};

  always @ (posedge clk) begin
    run <= rst & (reply | run); Q0 <= msclk; Q1 <= Q0;
    shreg <= ~rst ? InitBuf
      : (endbit | reply) ? -1 : shift ? {msdat, shreg[31:1]} : shreg;
    x <= ~rst ? 0 : endbit ? x + dx : x;  y <= ~rst ? 0 : endbit ? y + dy : y;
    btns <= ~rst ? 0 : endbit ? {shreg[1], shreg[3], shreg[2]} : btns;
  end

endmodule
```

VIDExample.0.Lola

```
MODULE VIDExample(  (*PDR 3.12.19*)
  IN OSCIN: BIT;  (*100MHz oscillator*)
  OUT hsync, vsync: BIT; RGB: [3] BIT);  (*VGA display*)

TYPE
  PLL = MODULE(IN refclkin: BIT; OUT globalclkout: BIT) ^;

VAR
  pll: PLL;
  clk, hend, vend, blank: BIT;

REG(clk)
  hcnt: [11] BIT; vcnt: [10] BIT; hsr, vsr: BIT;

BEGIN pll(OSCIN, clk);  (*100MHz -> 65MHz*)
  hend := (hcnt = 1344-1);  (*65MHz/1344 = 48.4KHz*)
  vend := (vcnt = 806-1);  (*48.4KHz/806 = 60.0Hz*)
  hcnt := hend -> 0 : hcnt + 1;
  vcnt := hend -> vend -> 0 : vcnt + 1 : vcnt;
  hsync := ~hsr; hsr := (hcnt = 1048) | hsr & (hcnt # 1184);
  vsync := ~vsr; vsr := (vcnt = 771) | vsr & (vcnt # 777);
  blank := hcnt.10 | (vcnt.9 & vcnt.8);
  RGB := blank -> 0 : hcnt[3:1] | vcnt[3:1];
END VIDExample.
```

VIDExample.1.Lola

```
MODULE VIDExample(  (*PDR 3.12.19*)
  IN OSCIN: BIT;  (*100MHz oscillator*)
  OUT hsync, vsync: BIT; RGB: [3] BIT);  (*VGA display*)

TYPE
  PLL = MODULE(IN refclkin: BIT; OUT globalclkout: BIT) ^;

VAR
  pll: PLL;
  col: [9] BIT;
  clk, hend, vend, blank, border, cursor: BIT;

REG(clk)
  hcnt: [11] BIT; vcnt: [10] BIT; fcnt: [9] BIT; hsr, vsr: BIT;

BEGIN pll(OSCIN, clk);  (*100MHz -> 65MHz*)
  hend := (hcnt = 1344-1);  (*65MHz/1344 = 48.4KHz*)
  vend := (vcnt = 806-1);  (*48.4KHz/806 = 60.0Hz*)
  hcnt := hend -> 0 : hcnt + 1;
  vcnt := hend -> vend -> 0 : vcnt + 1 : vcnt;
  fcnt := hend & vend -> fcnt + 1 : fcnt;
  hsync := ~hsr; hsr := (hcnt = 1048) | hsr & (hcnt # 1184);
  vsync := ~vsr; vsr := (vcnt = 771) | vsr & (vcnt # 777);
  blank := hcnt.10 | (vcnt.9 & vcnt.8);
  col := hcnt[9:1];
  border := (col = 0) | (col = 511) | (vcnt[9:1] = 0) | (vcnt[9:1] = 383);
  cursor := (col = fcnt);
  RGB := blank -> 0 : border -> 7 : {0'1, 0'1, cursor};
END VIDExample.
```

VIDExample.Lola

```
MODULE VIDExample(  (*PDR 3.12.19*)
  IN OSCIN: BIT;  (*100MHz oscillator*)
  OUT I2Sen, I2Sclk, I2Sws: BIT; IN I2Sdin: BIT;  (*I2S input*)
  OUT hsync, vsync: BIT; RGB: [3] BIT);  (*VGA display*)

TYPE
  PLL = MODULE(IN refclkin: BIT; OUT globalclkout: BIT) ^;
  BRAM512x8 = MODULE(IN wclk, rclk, wr, rd: BIT;
    IN wadr, radr: [9] BIT; IN wdata: BYTE; OUT rdata: BYTE) ^;

VAR
  pll: PLL; bram: BRAM512x8;
  clk, hend, vend, blank, border, pixel, display, cursor: BIT;
  value: BYTE; row, col: [9] BIT; capture: BIT;

REG(clk)
  hcnt: [11] BIT; vcnt: [10] BIT; fcnt: [9] BIT; hsr, vsr: BIT;
  scnt: [10] BIT; inshr: [18] BIT;

BEGIN pll(OSCIN, clk);  (*100MHz -> 65MHz*)
  hend := (hcnt = 1344-1);  (*65MHz/1344 = 48.4KHz*)
  vend := (vcnt = 806-1);  (*48.4KHz/806 = 60.0Hz*)
  hcnt := hend -> 0 : hcnt + 1;
  vcnt := hend -> vend -> 0 : vcnt + 1 : vcnt;
  fcnt := hend & vend -> fcnt + 1 : fcnt;
  hsync := ~hsr; hsr := (hcnt = 1048) | hsr & (hcnt # 1184);
  vsync := ~vsr; vsr := (vcnt = 771) | vsr & (vcnt # 777);
  blank := hcnt.10 | (vcnt.9 & vcnt.8);
  col := hcnt[9:1];
  border := (col = 0) | (col = 511) | (vcnt[9:1] = 0) | (vcnt[9:1] = 383);
  row := {vcnt.8 ^ vcnt.7, vcnt.7, ~vcnt[6:0]};
  display := vcnt.9 ^ (vcnt[8:7] # 0);
  cursor := (col = fcnt);
  RGB := blank -> 0
    : border -> 7 : pixel -> cursor -> 1 : {1'1, row.8, 0'1} : 0;
  scnt := scnt + 1; I2Sen := 1;
  I2Sclk := scnt.3; I2Sws := scnt.9; (*clk/16 = 4.0625MHz; I2Sclk/32*)
  inshr := (scnt[3:0] = 0FH) -> {inshr[16:0], I2Sdin} : inshr;
  capture := (scnt = 018FH);
  bram(clk, clk, capture, 1'1, fcnt, hcnt[9:1], inshr[17:10], value);
  pixel := display & (value >= row[8:1]);
END VIDExample.


./L2V VIDExample.Lola \
&& yosys -q -p 'synth_ice40 -blif videxample.blif' \
  VIDExample.Lola.v PLL.v BRAM512x8.v  \
&& arachne-pnr -d 8k -P tq144:4k -o videxample.asc -p VIDExample.pcf
videxample.blif \
&& icepack videxample.asc videxample.dfu && dfu-suffix -a videxample.dfu \
&& dfu-util -D videxample.dfu

icetime -d hx8k -P tq144:4k -p VIDExample.pcf videxample.asc
```