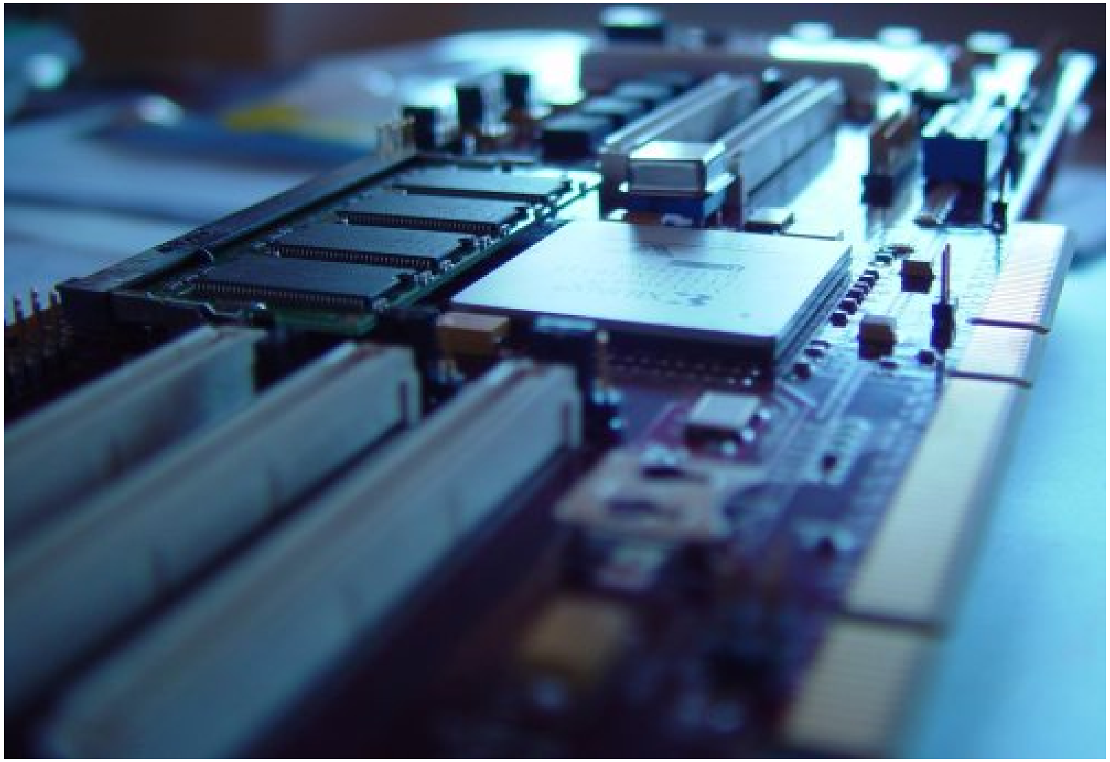
TSEA44: Computer hardware – a system on a chip

”dafk”
”tsea21”
”tsea02”



Andreas Ehliar, Andreas Karlsson, Kent Palmqvist
http://www.da.isy.liu.se/courses/tsea44/

What is the course about?

- How to build a complete embedded computer using an FPGA and a few other components. Why?
- Only one chip
- The computer can easily be tailored to your needs.
  - Special instructions
  - Accelerators
  - DMA transfer
- The computer can be simulated
- A logic analyzer can be added in the FPGA
  - Add performance counters
- It’s fun!

Prerequisites

You will definitely need a thorough understanding of

* ***Digital logic design.*** You will design both a data path and a control unit for an accelerator.

* ***Binary arithmetic.*** Signed/unsigned numbers.

* ***VHDL or Verilog.*** SystemVerilog (SV) is the language used in the course.

* ***Computer Architecture.*** It is extremely important to understand how a CPU executes code. You will also design part of a DMA-controller. Bus cycles are central.

* ***Asm and C programming.*** Most of the programming is done in C, with a few cases of inline asm.

Course organisation

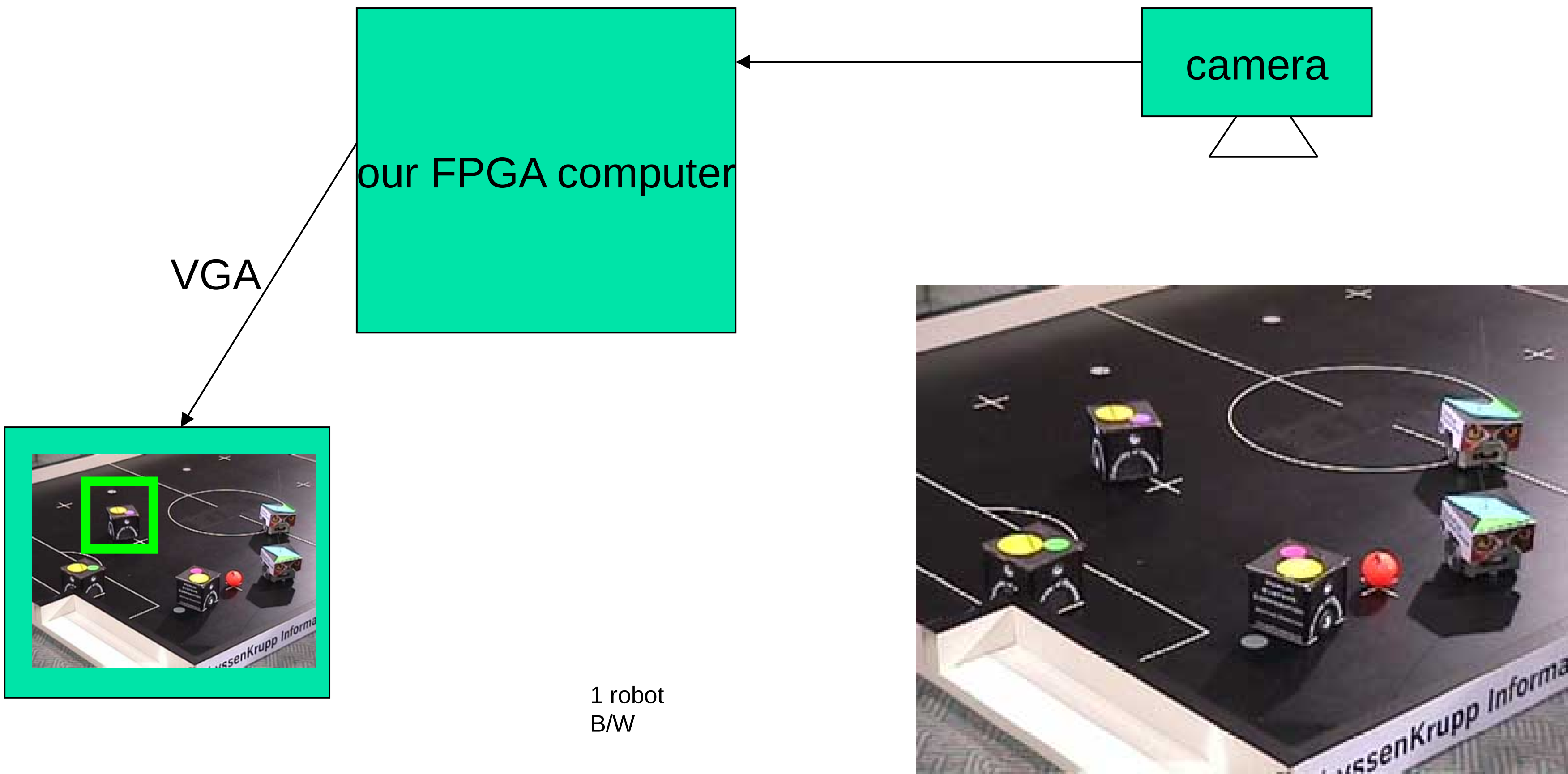
- Lab course: 4 mini projects
  - 6 groups * 3 students in the lab
- Lab 0: learn enough Verilog, 4 hours

- Lectures: 8*2 hours

- Examination 6CPs:
  - 3 written reports/group
  - oral individual questions

The lab course is based on an application
2004 - tracking

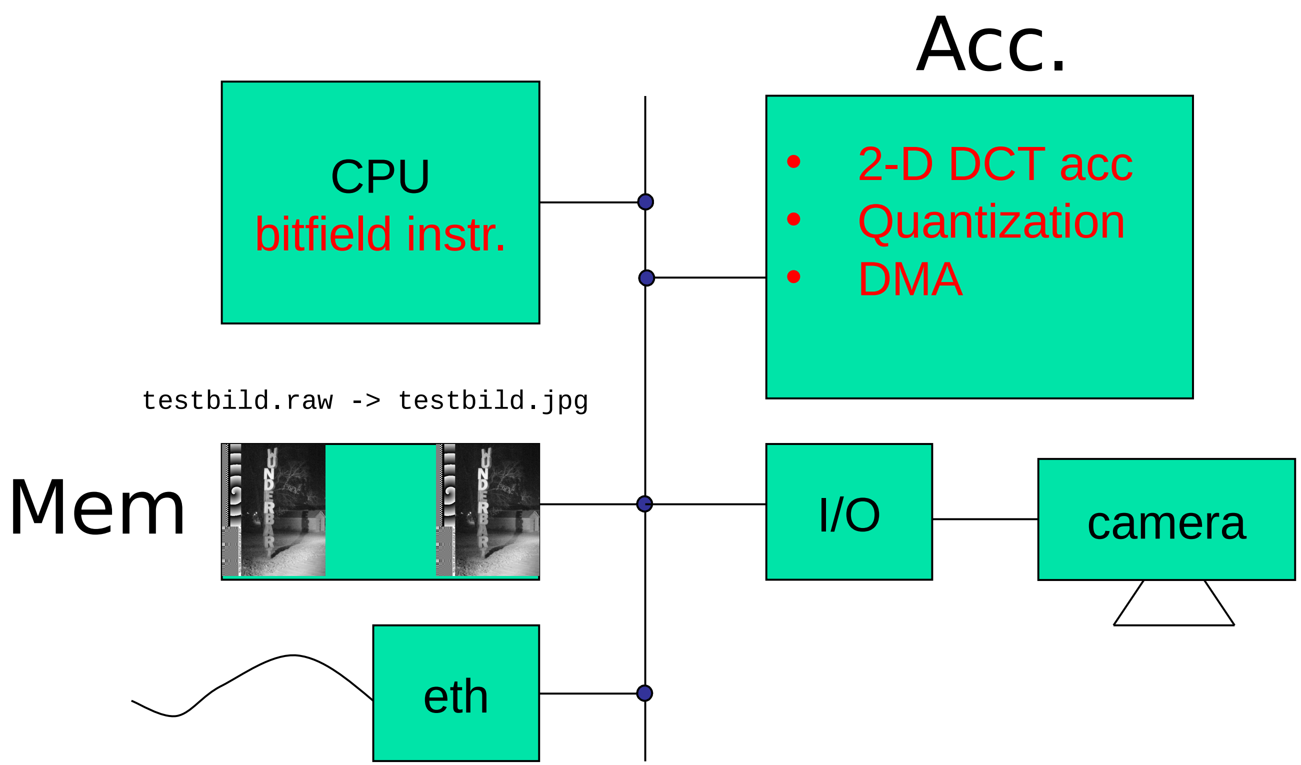
The application is inspired from a robot football web page:
`http://www.tech.plym.ac.uk/robofoot/`



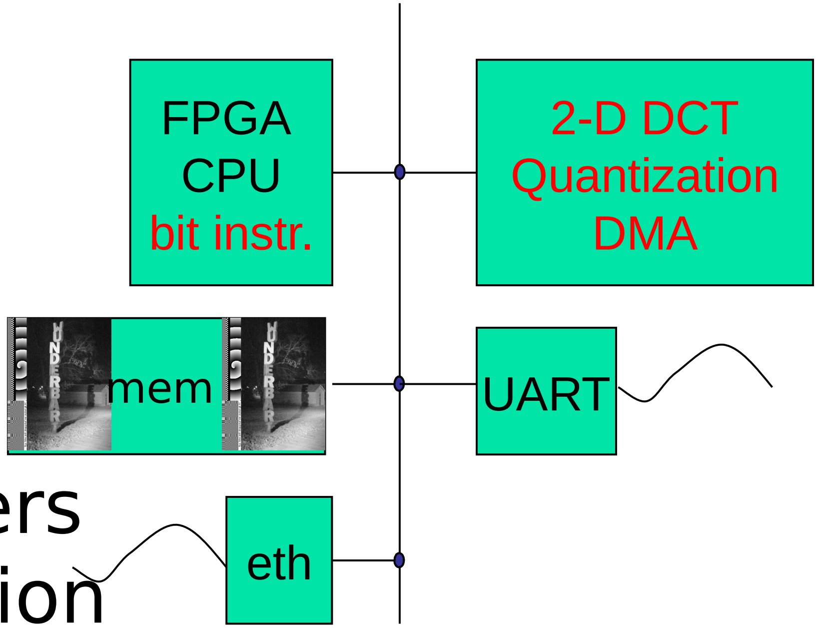
The lab course is based on an application

2005-12 – JPEG compression

- Take 2-D DCT on 8x8-blocks
- Quantize = Divide and set small values to zero
- RLE + Huffman code



Lab info



- 0) Build a UART in Verilog

- 1) Interface your UART
  Test performance counters
  Test a SW-DCT2 application

- **2+3) Build a HW accelerator for 2-D DCT and add a DMA controller**

- **4) Design your own instruction to handle bit fields**

•Lab 0 should be done on an *individual* basis
•Your group shall send in 3 written lab reports in PDF format (via Urkund).

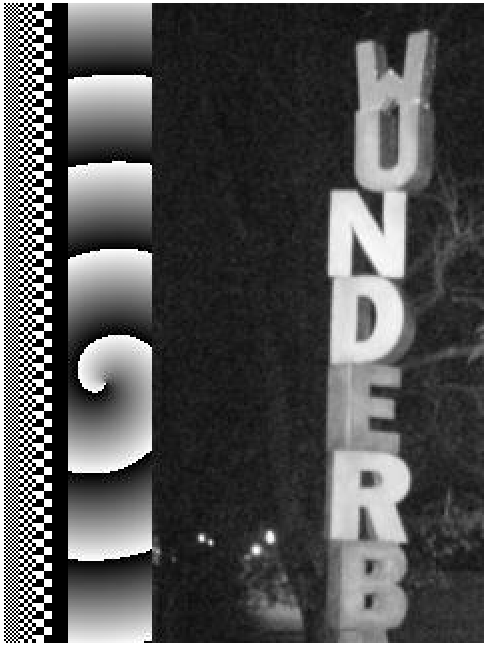
Lab nr	**Lab task**	**Examination**
0	Build a UART in Verilog	Demonstration
1	Interfacing to the Wishbone bus	Demonstration Written report
2+3	Design a JPEG accelerator + DMA	Demonstration Written report
4	Custom Instruction	Demonstration Written report

Demonstration = presentation of working design. We ask individual questions!

Written report = a readable short report typically consisting of
- **Introduction**
- **Design**, where you explain with text and diagrams how your design works
- **Results**, that you have measured
- **Conclusions**
- **Appendix** : All Verilog and C code with comments!

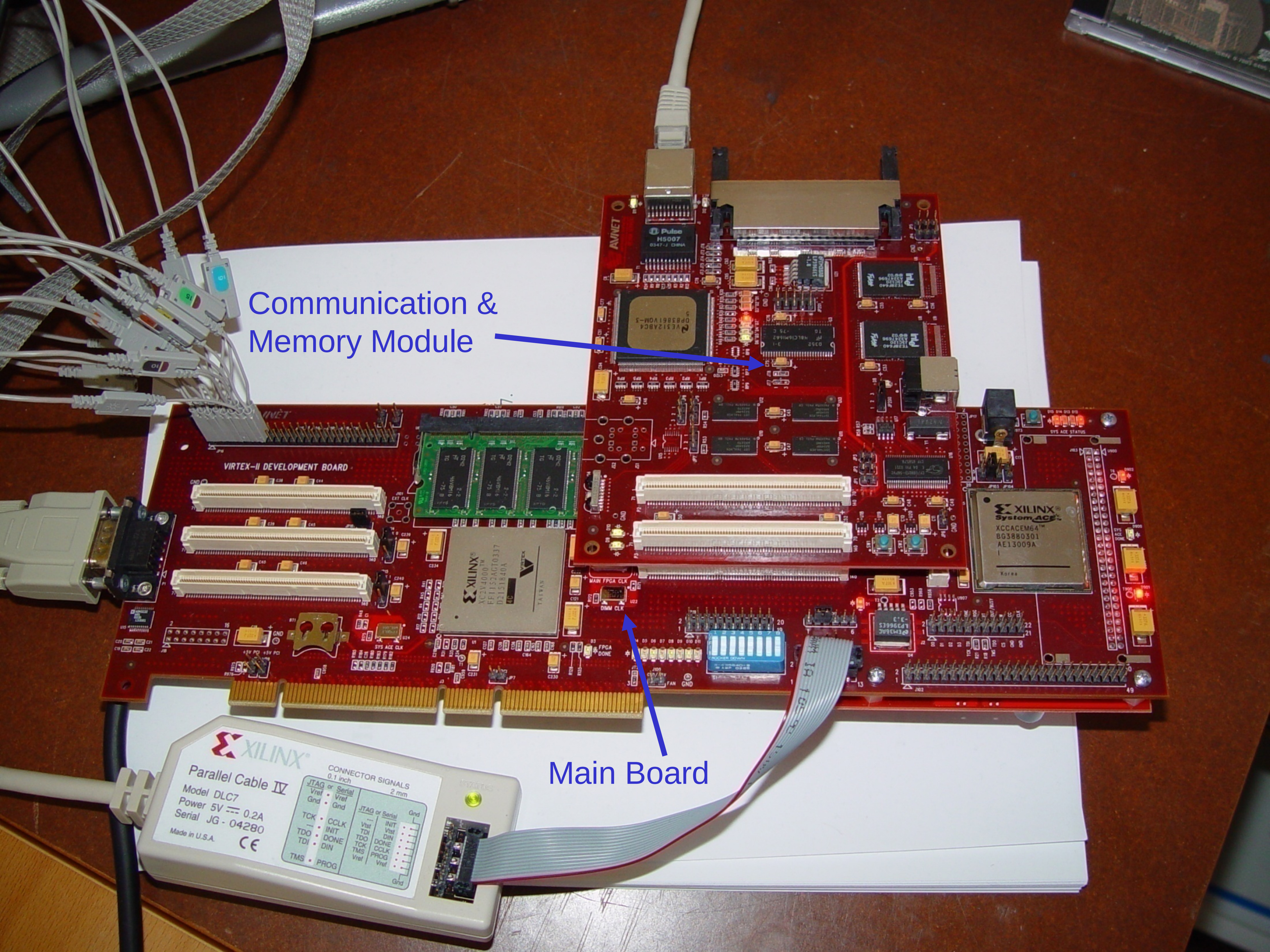
Competition – fastest JPEG compression

- An unaccelerated JPEG compression (using `jpegfiles`) takes roughly 13,0 Mcycles (25 Mhz).
- Our record: ~100000 cycles (everything in hardware at this point).
- Goal: Highest frame rate. Exception: At over 25 FPS, the smallest implementation wins.

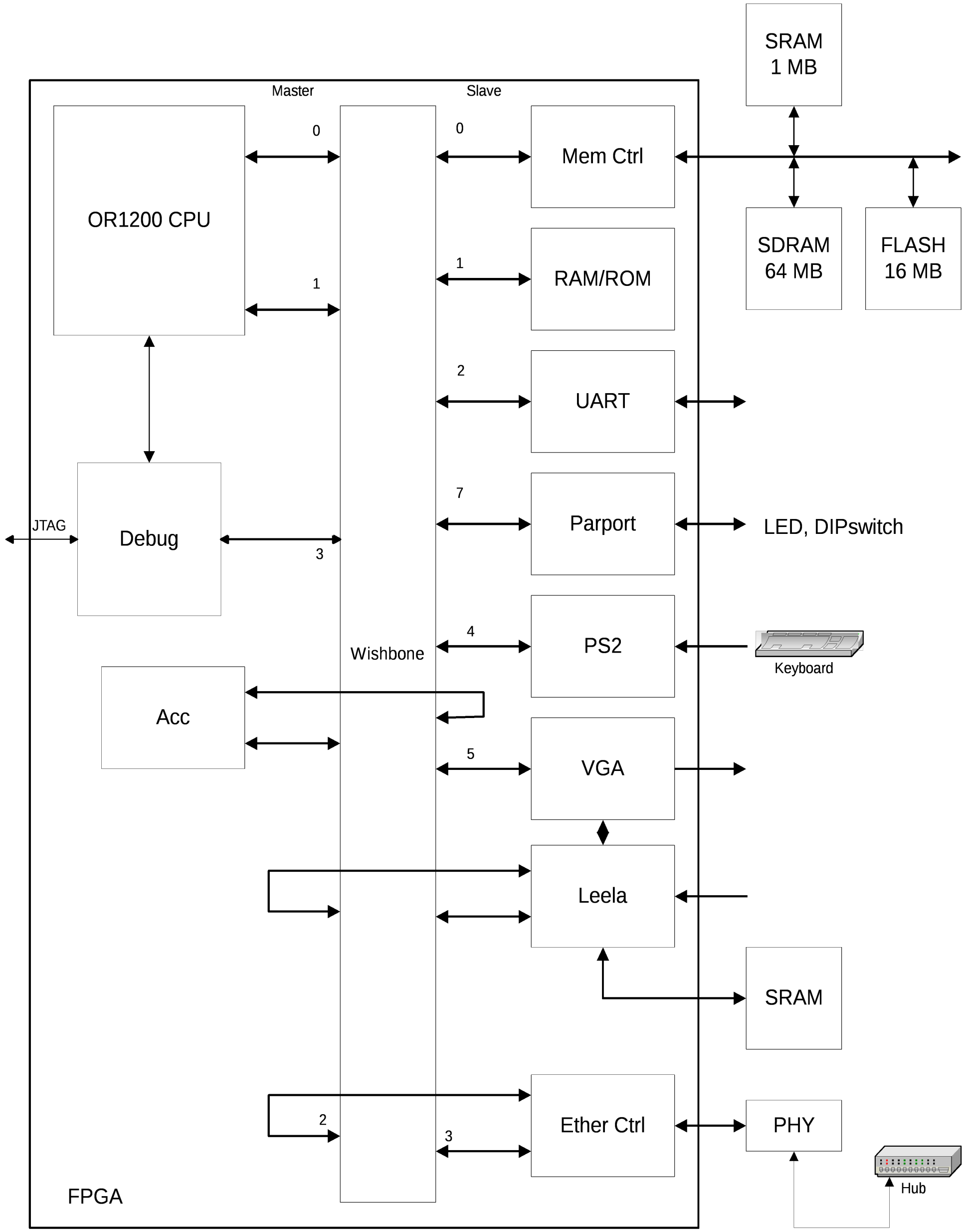


wunderb.jpg
320 x 240

Wunderbart-tårtan


Communication &
Memory Module
Main Board
VIRTEX-II DEVELOPMENT BOARD
XILINX
System ACE
XILINX
Parallel Cable IV
Model DLC7
Power 5V 0.2A
Serial JG - 04280
Made in U.S.A.
CONNECTOR SIGNALS

Our "soft" computer



OPENCORES.ORG

OPENCORES.ORG
free open source IP cores and chip design

printer

LOGIN ::: RECOVER PASS ::: GET AC(

Browse

- Projects
- CVS
- Forums
- News
- Articles
- Polls

Opencores

- FAQ
- CVS HowTo
- Mission
- Media
- Tools
- Sponsors
- Mirrors
- Logos
- Contact us

Tools

- Search
- CVSGet

More

By category :: Last updated :: Last created :: Most popular :: Best

Projects by category

We use a few icons to help identify projects:

new Indicates new project, that has been added in the last 30 da
done Indicates project that is ready to use
wbc Indicates a WISHBONE Compliant Core

Click on category to see only its projects

Note: language filter doesn't work very well yet because most of projects don't have this property set. We are asking developers to set it.

Projects/cores: Ready to be used and Wishbone compliant and written in any language Show

Arithmetic core

- CORDIC core done
- 5x4Gbps CRC generator designed with standard cells done
- Single Clock Unsigned Division Algorithm done

Microprocessor

- Mini-Risc core done
- Plasma - most MIPS I(TM) opcodes done
- OpenRISC 1000 wbc don
- Yellow Star done

(System)Verilog!

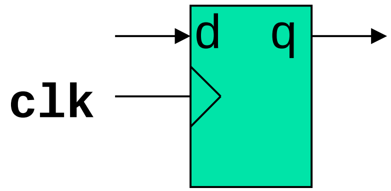
- The course uses SystemVerilog!
- SystemVerilog is easy to learn if you know VHDL/C
- Our soft computer (80 % downloaded from OpenCores) is written in Verilog
- It is possible to use both languages in a design
- You need to understand parts of the computer

SystemVerilog vs VHDL

an edge-triggered D-flip/flop

C-like syntax

```
module dff(
  input clk, d,
  output reg q);

  always_ff @(posedge clk)
   q <= d;

endmodule
```

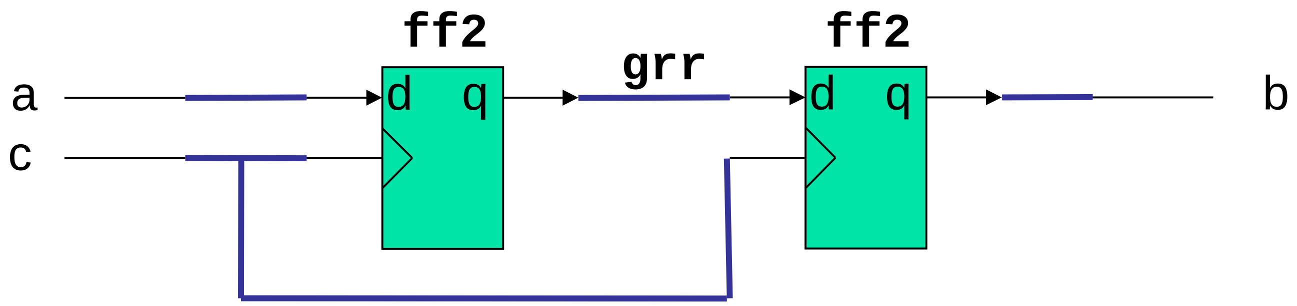


Ada-like syntax

```
entity dff is
port (clk,d : in std_logic;
      q: out std_logic);
end dff;

architecture firsttry of dff is
begin
process (clk) begin
 if rising_edge(clk) then
   q <= d;
 end if;
end process;
end firsttry;
```

SystemVerilog vs VHDL

let's use our D-flip/flop, instantiation

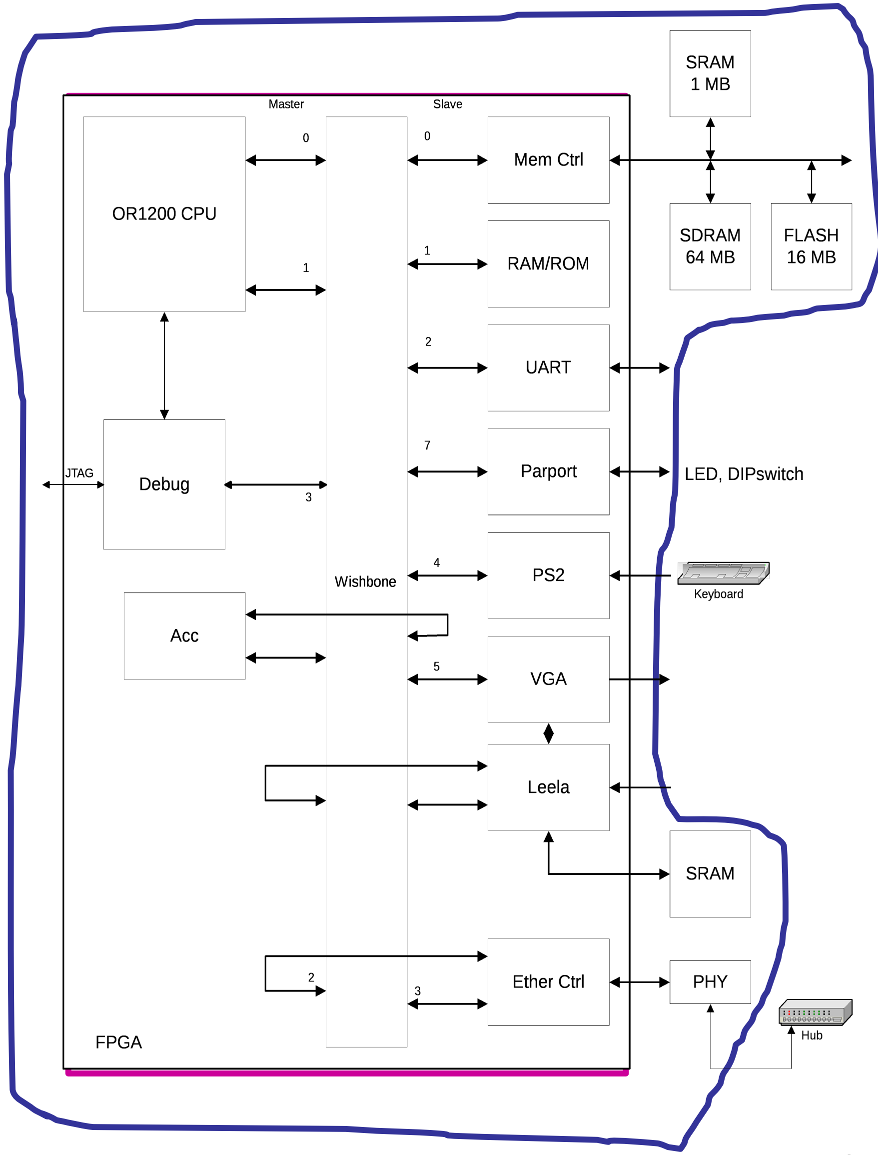



```
// instantiation

wire a,b,c,grr;
...

dff ff1(.clk(c),.d(a), .q(grr));

dff ff2(.clk(c), .d(grrr), .q(b));
```

Watch out! Verilog allows implicit declarations (but this can be disabled)

You get a lab skeleton!

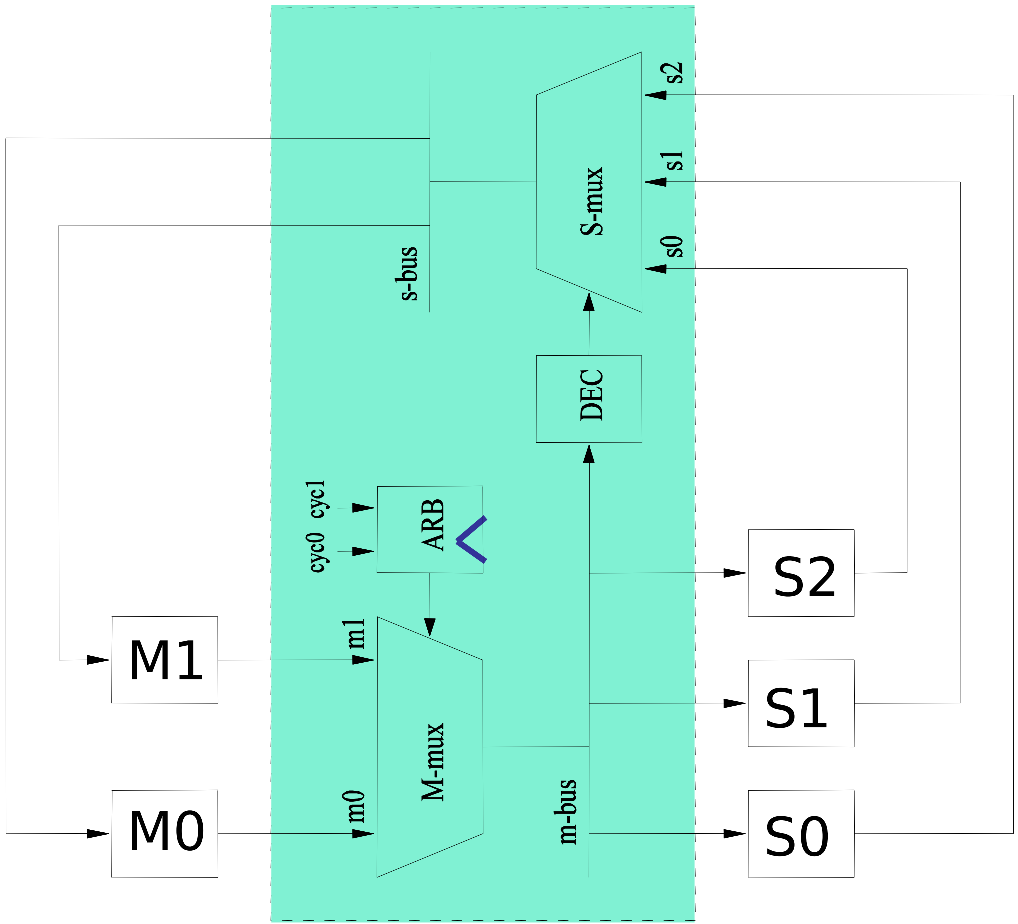
- **dafk_tb.sv** . Testbench.
  - **dafk_top.sv** . To be synthesized in the FPGA.
    - . **eth_top.sv**. Ethernet controller.
    - . **pkmc_top.sv**. Memory controller.
    - . **or1200_top.sv**. The OR1200 CPU.
    - . **parport.sv**. Simple parallel port.
    - . **romram.sv** . The boot code resides here.
    - . **uart_top.sv** . UART 16550.
    - . **dvga_top.sv** . VGA controller.
    - . **wb_top.sv** . The wishbone bus.
  - **eth_phy.v** . Simulation model for the PHY chip.
  - **flash.v** Simulation model.
  - **sdram.v** Simulation model.
  - **sram.v** . Simulation model.

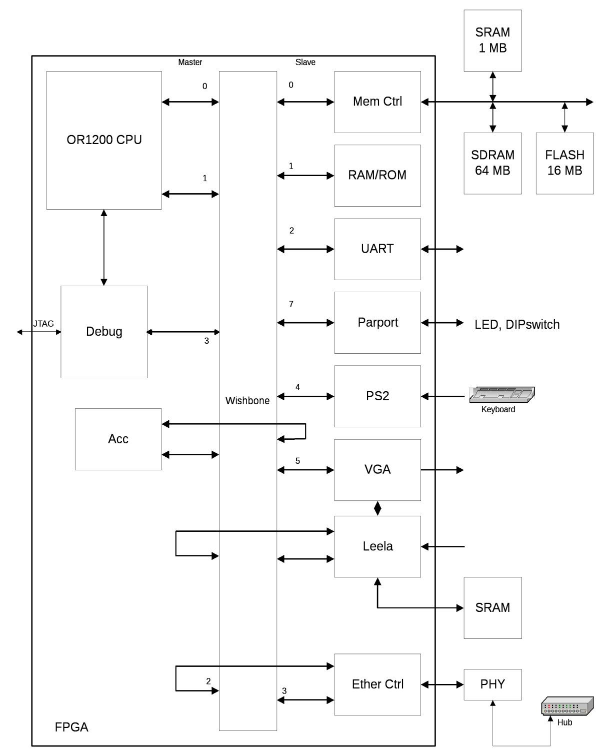


The Wishbone bus

A multi-master bus

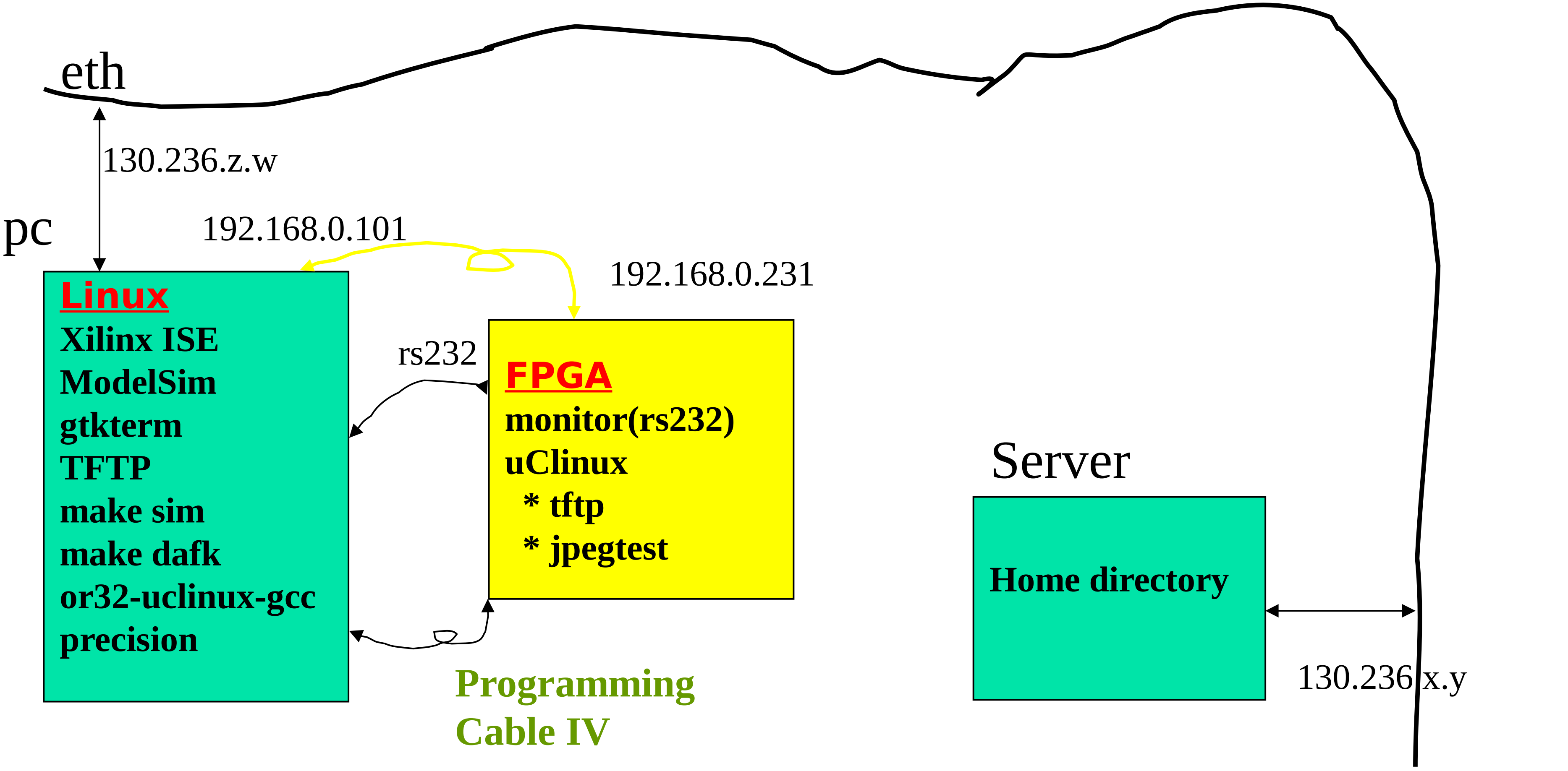
- Signals: adress (32), data_out(32), data_in(32), control
- Two data buses and muxes are used instead of tristate





"The environment"

- We prefer linux (centos) but you can also use windows
  - Compile uClinux only on linux



Software under linux

- C-compiler (GNU tool chain)
  - `or32-uclinux-gcc`
- Software simulator
  - `or32-uclinux-sim`
- a very simple boot monitor (24 kB ROM + 8 kB RAM inside FPGA)
  - `dct_sw, dma_dct_sw,jpegtest`
- μClinux boots from flash
  - `jpegtest`

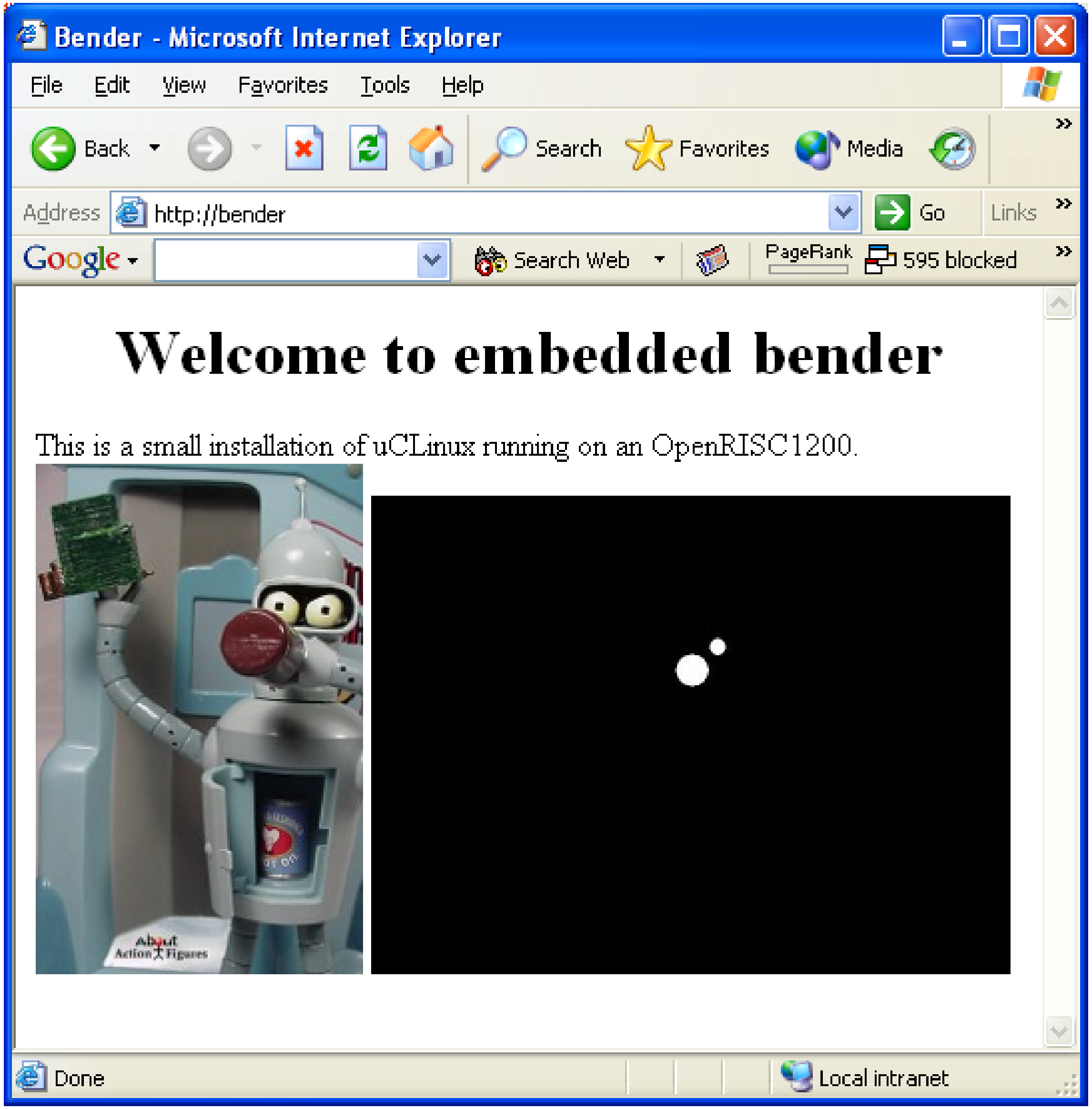
host

board

booting uClinux

```
uClinux/OR32
Flat model support (C) 1998,1999 Kenneth Albanowski, D. Jeff Dionne
Calibrating delay loop.. ok - 2.00 BogoMIPS
Memory available: 53000k/62325k RAM, 0k/0k ROM (667892k kernel data,
2182k code)
Swansea University Computer Society NET3.035 for Linux 2.0
NET3: Unix domain sockets 0.13 for Linux NET3.035.
Swansea University Computer Society TCP/IP for NET3.034
IP Protocols: ICMP, UDP, TCP
uClinux version 2.0.38.1pre3 (olles@kotte) (gcc version 3.2.3) #180
Sat Sep 11 0
9:01:55 CEST 2004
Serial driver version 4.13p1 with no serial options enabled
ttyS00 at 0x90000000 (irq = 2) is a 16550A
Ramdisk driver initialized : 16 ramdisks of 2048K size
Blkmem copyright 1998,1999 D. Jeff Dionne
Blkmem copyright 1998 Kenneth Albanowski
Blkmem 0 disk images:
loop: registered device at major 7
eth0: Open Ethernet Core Version 1.0
RAMDISK: Romfs filesystem found at block 0
RAMDISK: Loading 1608 blocks into ram disk... done.
VFS: Mounted root (romfs filesystem).
Executing shell ...
Shell invoked to run file: /etc/rc
Command: #!/bin/sh
Command: setenv PATH /bin:/sbin:/usr/bin
Command: hostname bender
Command: #
Command: mount -t proc none /proc
... More of the same
Command: #
Command: # start web server
Command: /sbin/boa -d &
[12]
/>
```

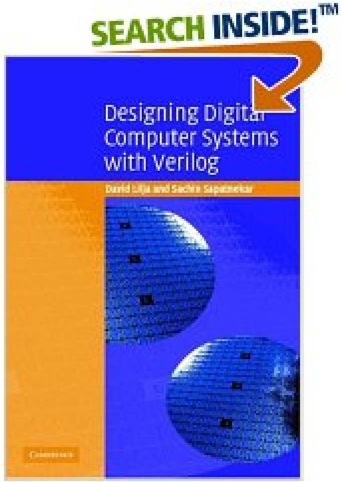
Web server


Bender - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Media
Address
http://bender
Go
Links
Google
Search Web
PageRank
595 blocked
Welcome to embedded bender
This is a small installation of uCLinux running on an OpenRISC1200.
Done
Local intranet

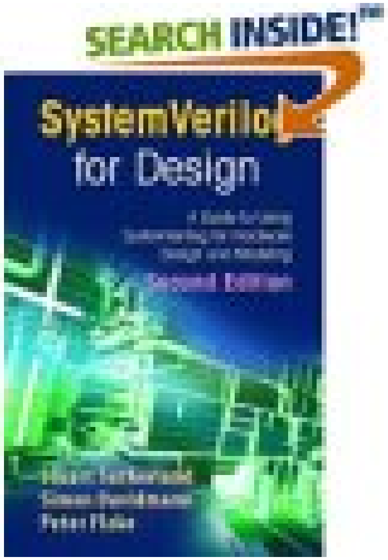
Lecture info

1. Course Intro, FPGA
2. Verilog (lab0)
3. A soft CPU
4. A soft computer (lab1)
5. HW Acceleration (lab2)
6. FPGAs
7. Test benches, SV
8. Custom instructions (lab4)

Books



Lilja,Saptnekar: *Designing Digital Computer Systems with Verilog*, Cambridge University Press

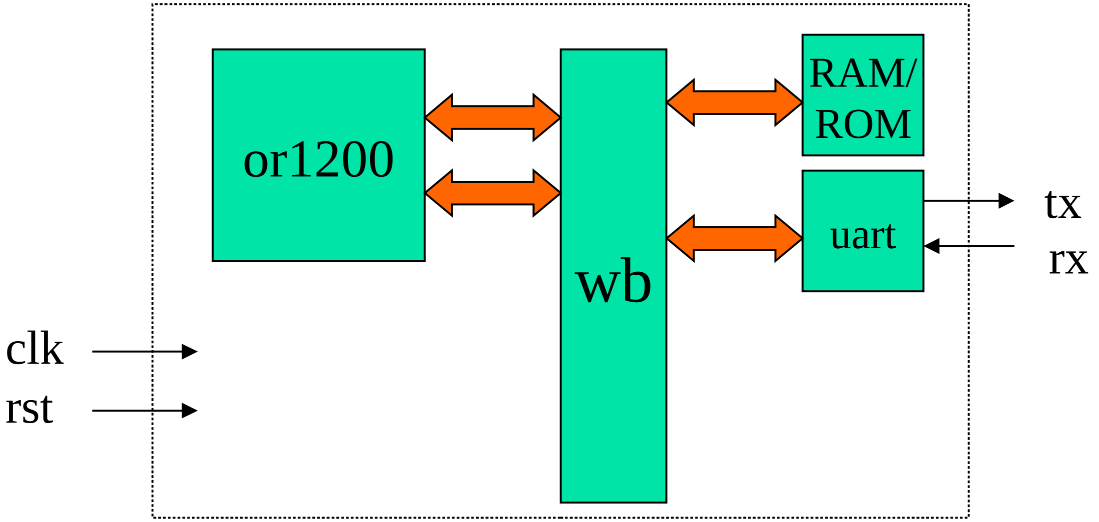


Sutherland et al: *SystemVerilog for Design*, Springer

Spear: *SystemVerilog for Verification*, Springer

How we built our first FPGA computer

1) Download CPU OR1200, roughly 60 Verilog files (tar/svn)
2) Download Wishbone bus 3 Verilog files
3) Download UART 16550, 9 Verilog files
4) Figure out a computer



How I built my first FPGA computer

5) Write top file (”wire wrap in emacs”)
   Size 35kB in Verilog, 13 kB in SV
   (Verilog does not have struct)

```
module myfirstcomputer(clk,rst,rx,tx)
  input clk,rst,rx;
  output tx;

  wishbone Mx[0:1], Sx[0:1];

  or1200cpu cpu0(.iwb(Mx[0]), … );
  wb_conbus wb0(clk, rst, Mx, Sx);
  romram rom0(Sx[1]);
  uart uart0(Sx[0], …);
end module
```

How I built my first FPGA computer

6) Download cross compiler

7) Write a small monitor and place in ROM

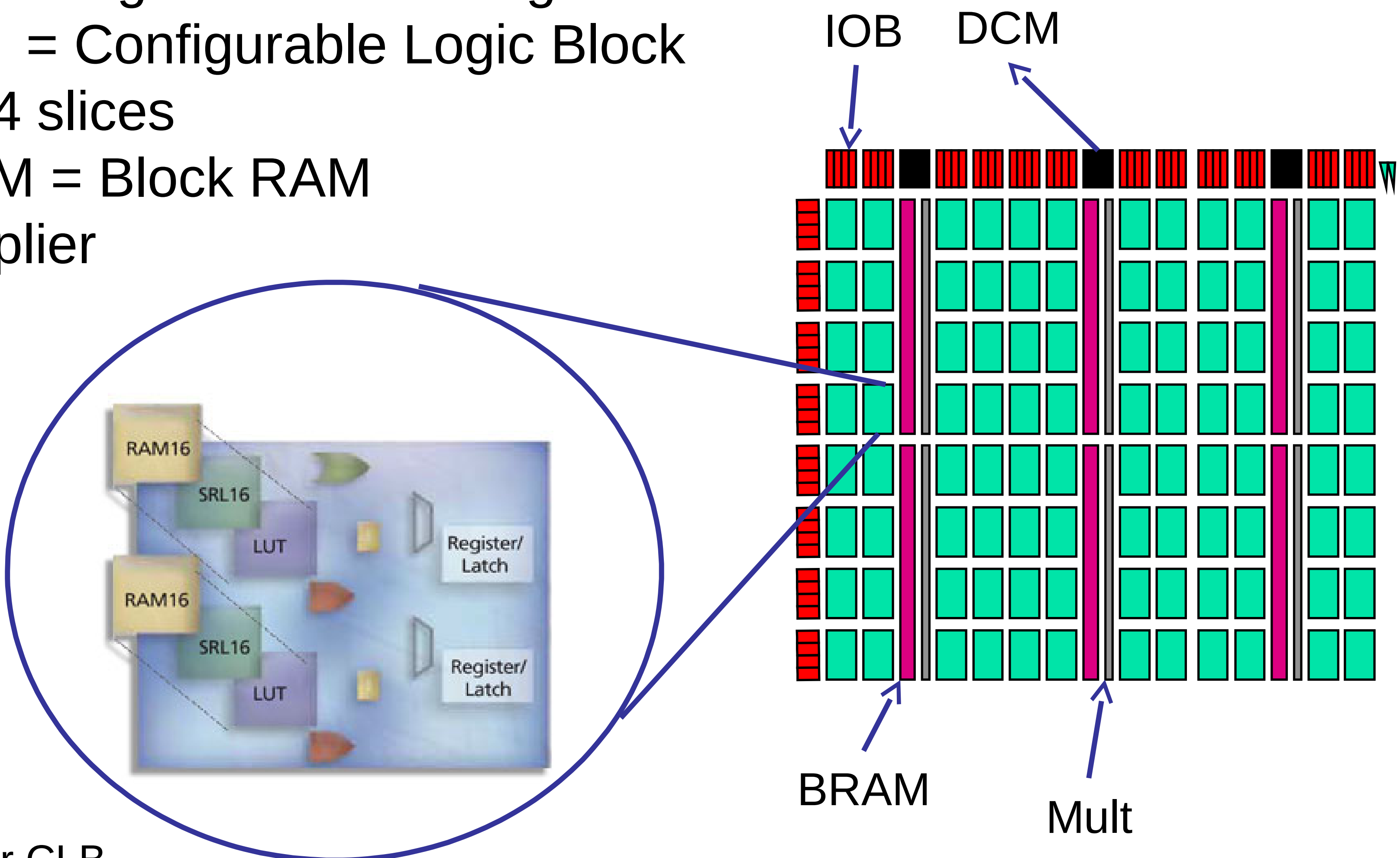
8) ModelSim. Does it boot? Anything on tx?

9) Test with the simulator or32-uclinux-sim

10) Synthesize for 10 min (originally 40 minutes, note that simulations are quite important in this course)

Xilinx – Virtex II Overview

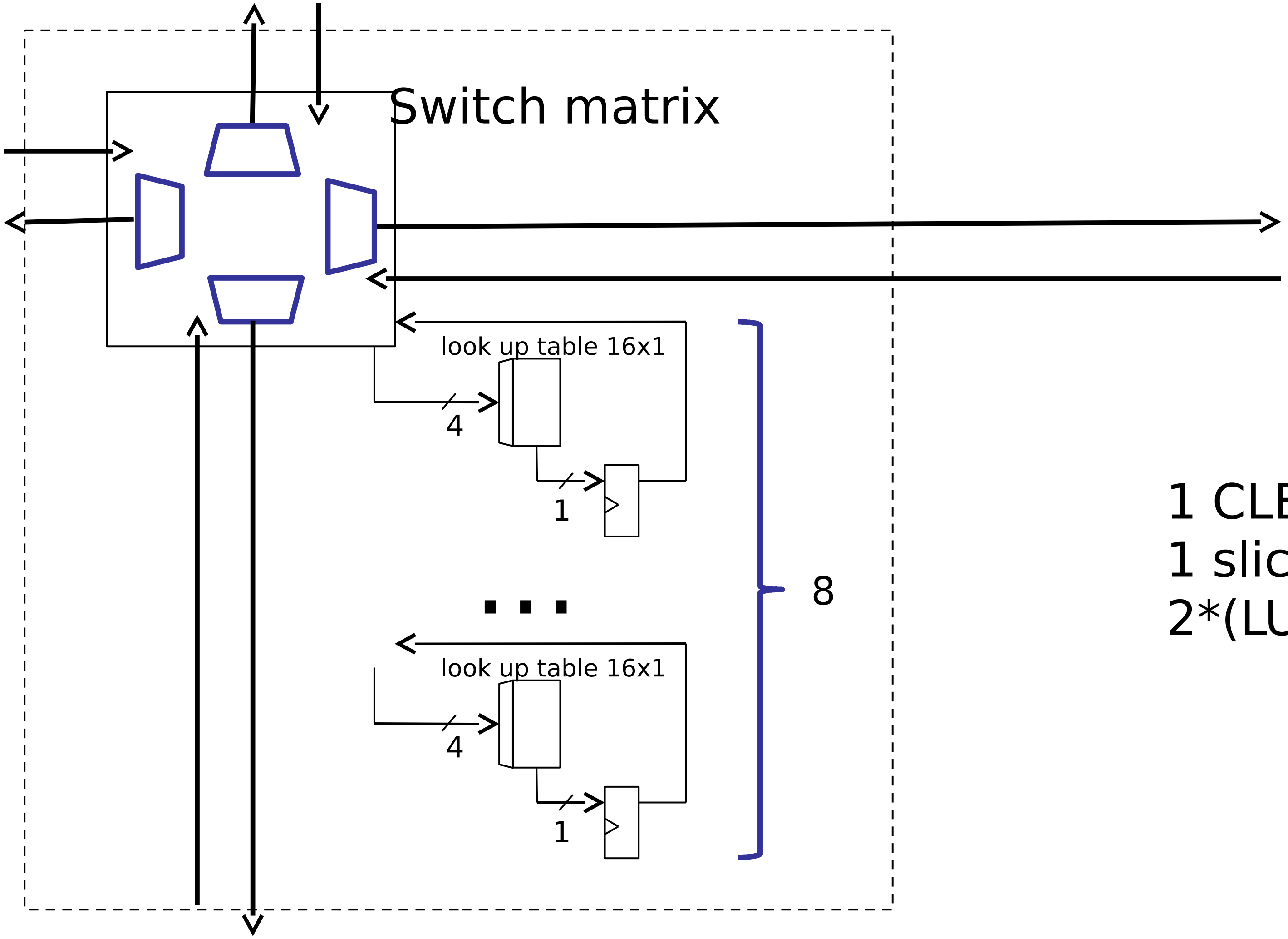
- IOB = I/O-block
- DCM = Digital Clock Manager
- CLB = Configurable Logic Block
  = 4 slices
- BRAM = Block RAM
- Multiplier



4 Slices per CLB.
1 slice = two F/F + two 4-input LUT

CLB = configurable logic block

LUT = look up table



1 CLB = 4 slices
1 slice = 2*(LUT+FF)

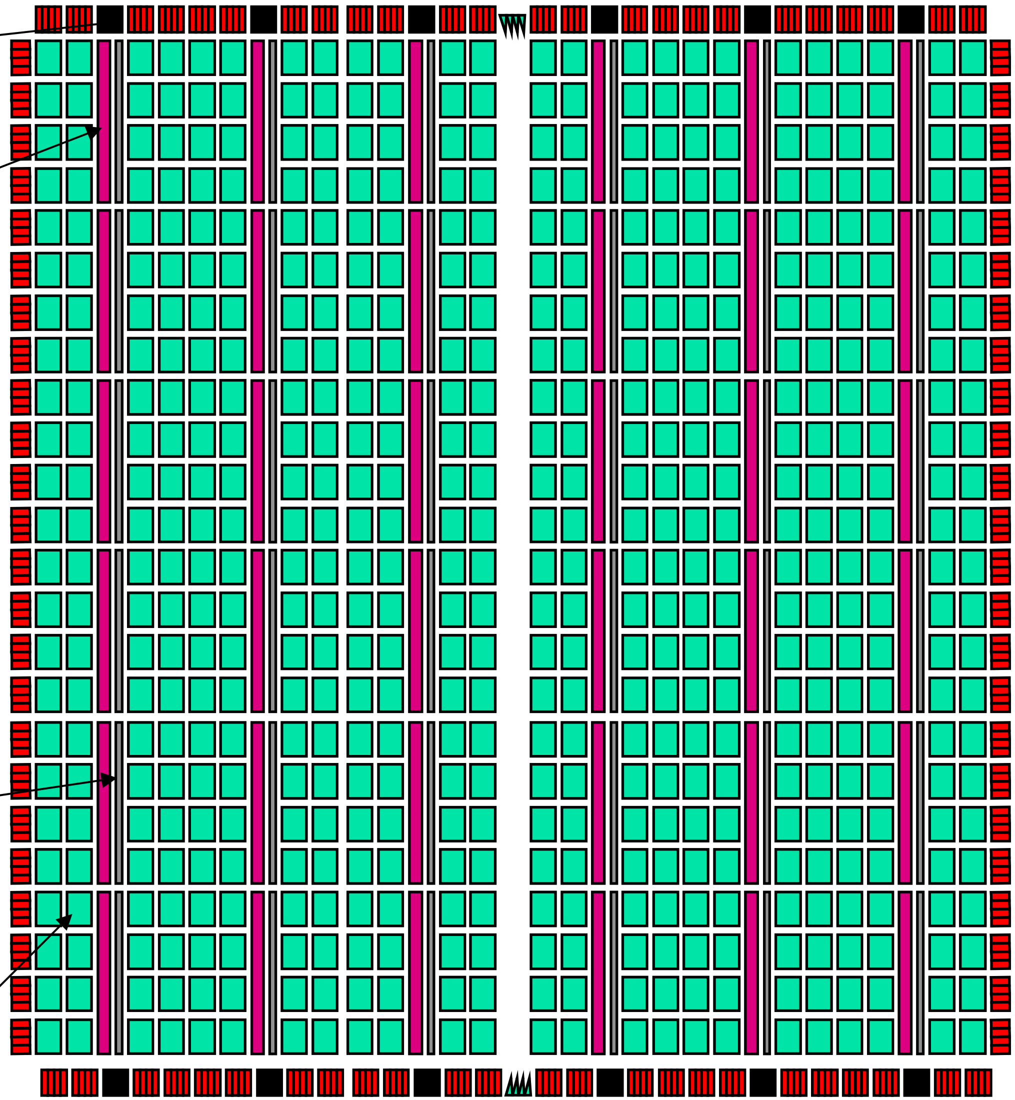
Our FPGA

DCM =
Digital Clock Manager (12)

block RAM (120)

IOB = I/O Block (912)

18x18 multiplier
(120)



CLB = configurable logic block (80x72=5760) => 46080 LUT/FF

Xilinx – Virtex II Overview

Device XC2V	40	80	250	500	1000	1500	2000	3000	4000	6000	8000
CLB Array	8 x 8	16 x 8	24 x 16	32 x 24	40 x 32	48 x 40	56 x 48	64 x 56	80 x 72	96 x 88	112 x 104
18Kb BRAM	4	8	24	32	40	48	56	96	120	144	168
Multiplier	4	8	24	32	40	48	56	96	120	144	168
DCM	4	4	8	8	8	8	8	12	12	12	12
Max IOB	88	120	200	264	432	528	624	720	912	1,104	1,296

2 Columns BRAM & Multipliers (40, 80)

4 Columns BRAM & Multipliers (250 – 2000)

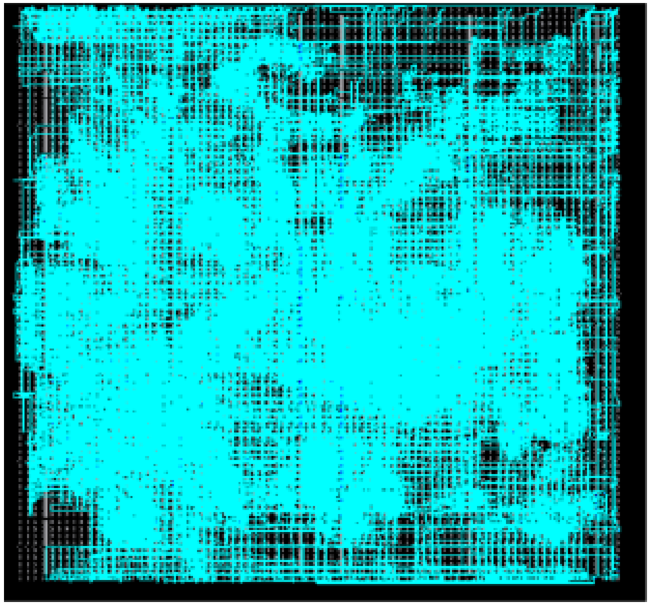
6 Columns BRAM & Multipliers (3000 – 8000)

Our FPGA has 5760 CLBs = 23.040 slices = 46080 LUTs+FFs

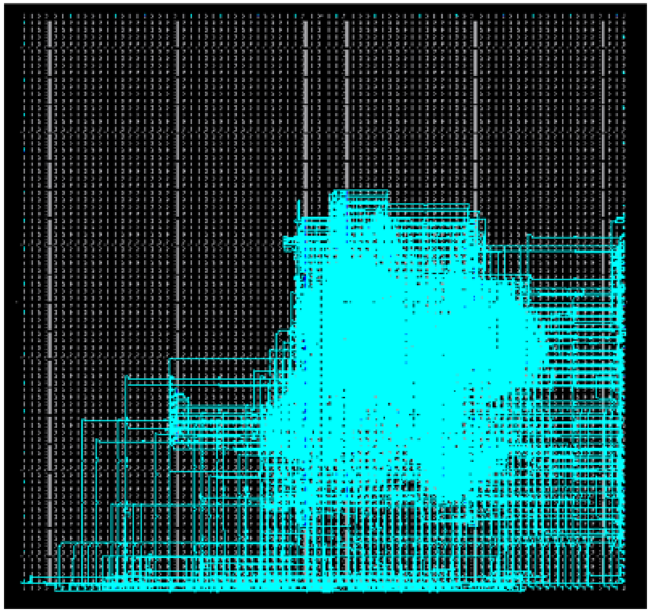
Synthesis result

Module	LUT	FF	RAMB16	MULT_18x18	IOB
/	64				216
cpu	5029	1345	12	4	
dvga	813	755	4		
eth3	3022	2337	4		
jpg0	2203	900	2	13	
leela	685	552	4	2	
pia	2	5			
pkmc_mc	218	122			
rom0	82	3	12		
sys_sig_gen		6			
uart2	825	346			
wb_conbus	616	11			
Total	**13559**	**6382**	**38**	**19**	**216**
Available	46080	46080	120	120	912

Floorplan from FPGA Editor



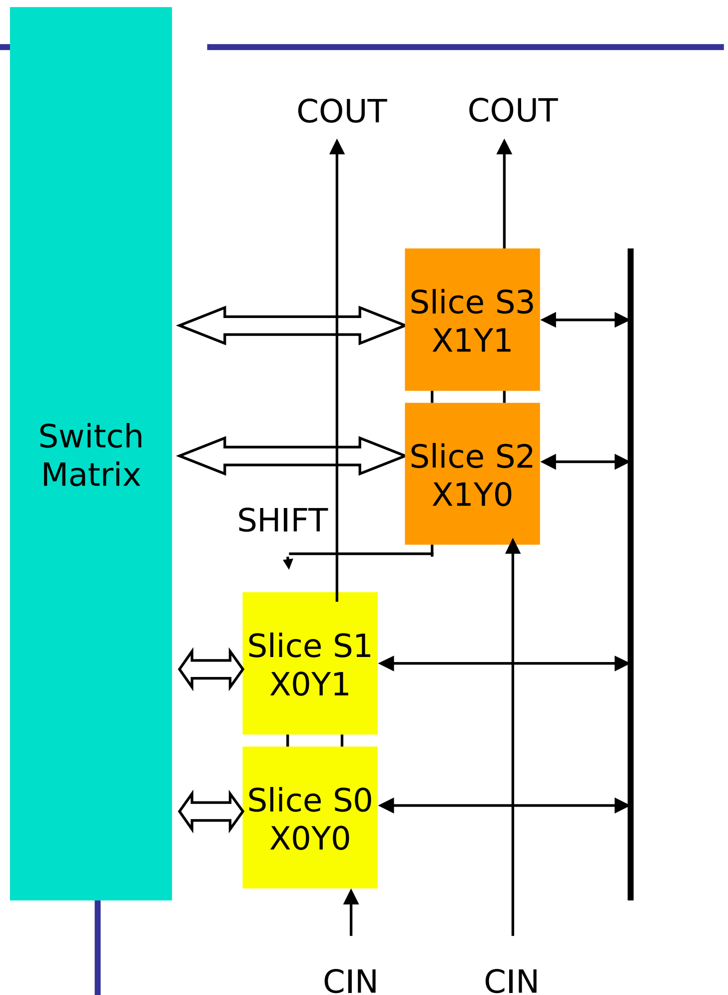
Computer



CPU OR1200

CLB Contains Four Slices

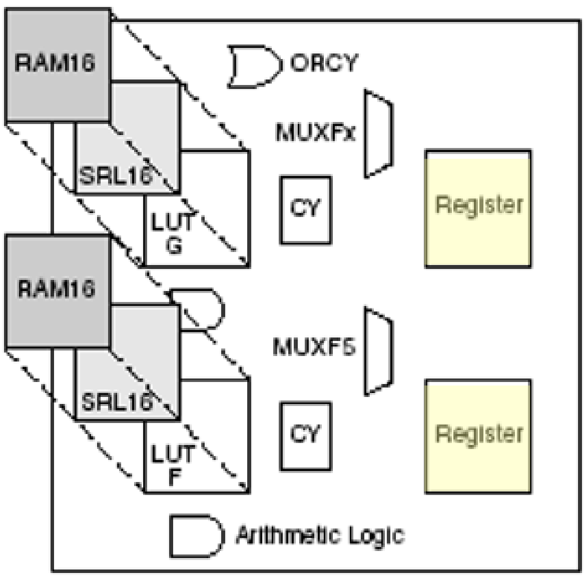
- Each CLB is connected to one switch matrix
  - 1 slice = 2 LUT/FF + ...

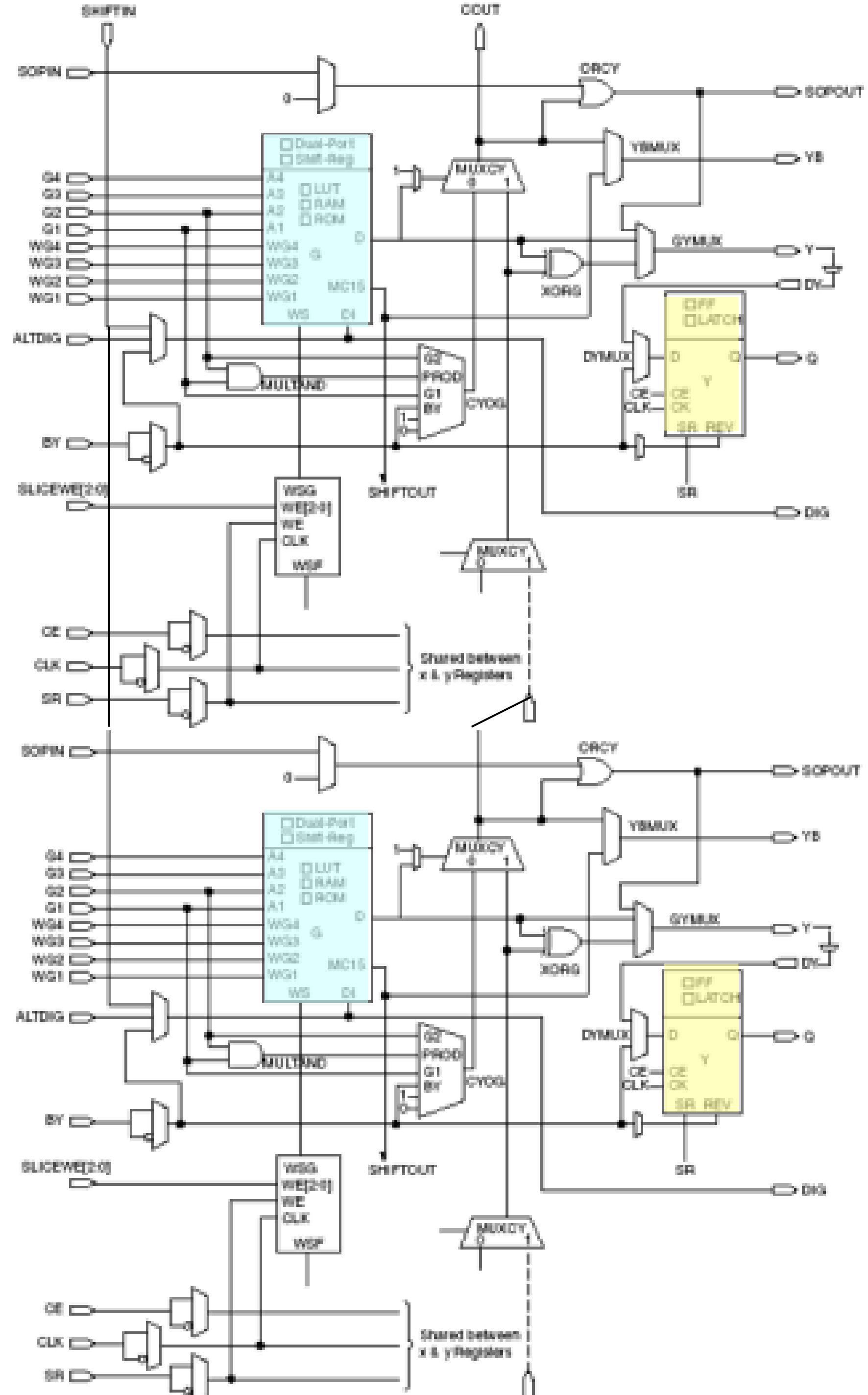


High level of logic integration

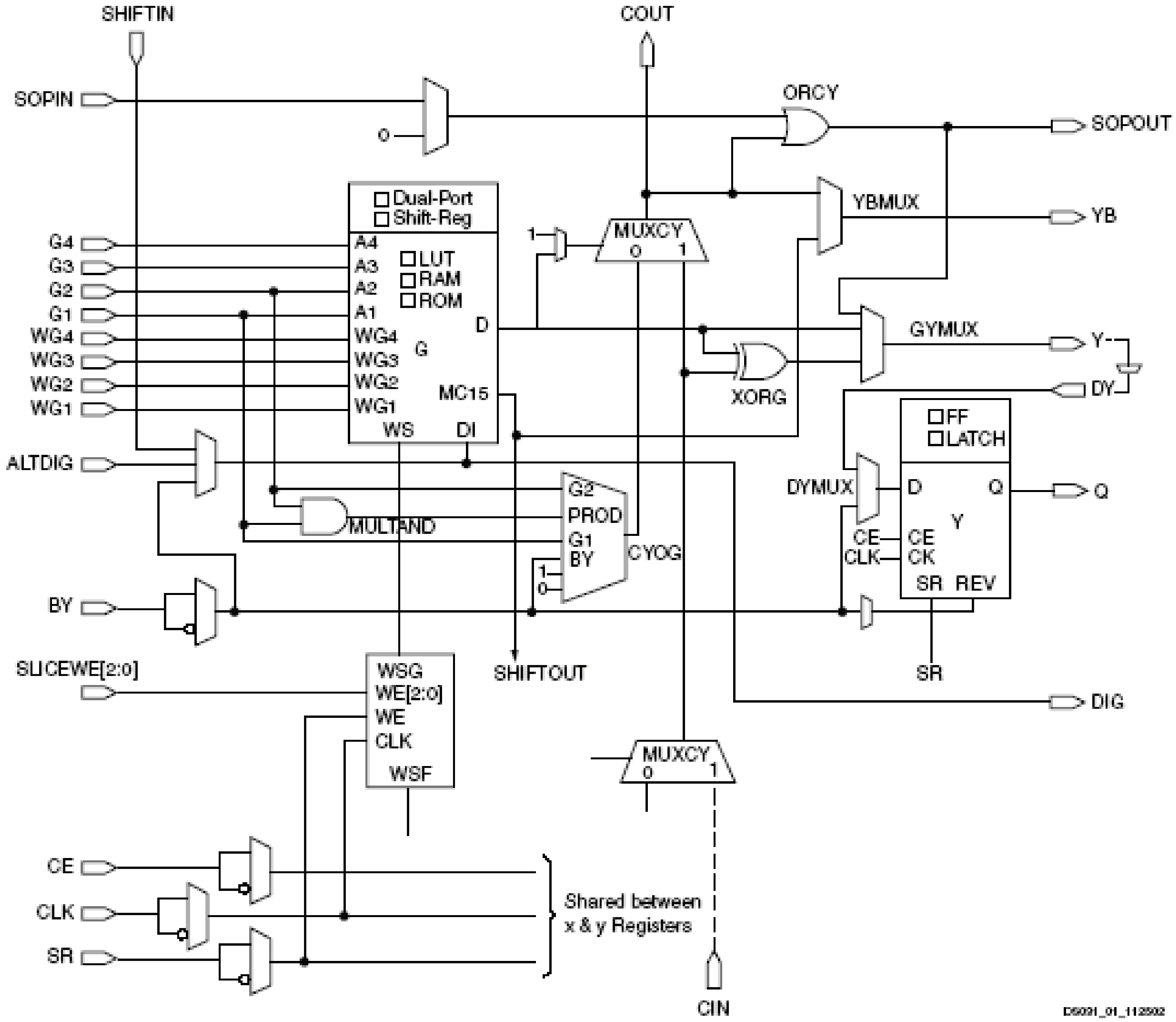
- Wide-input functions:
  - 16:1 multiplexer in 1 CLB
  - 32:1 multiplixer in 2 CLBs (1 level of LUT)
- Fast arithmetic functions
  - 2 look-ahead carry chains per CLB column
- Addressable shift registers in LUT
  - 16-b shift register in 1 LUT
  - 128-b shift register in 1 CLB (dedicated shift chain)

1 slice
(out of 23.040)

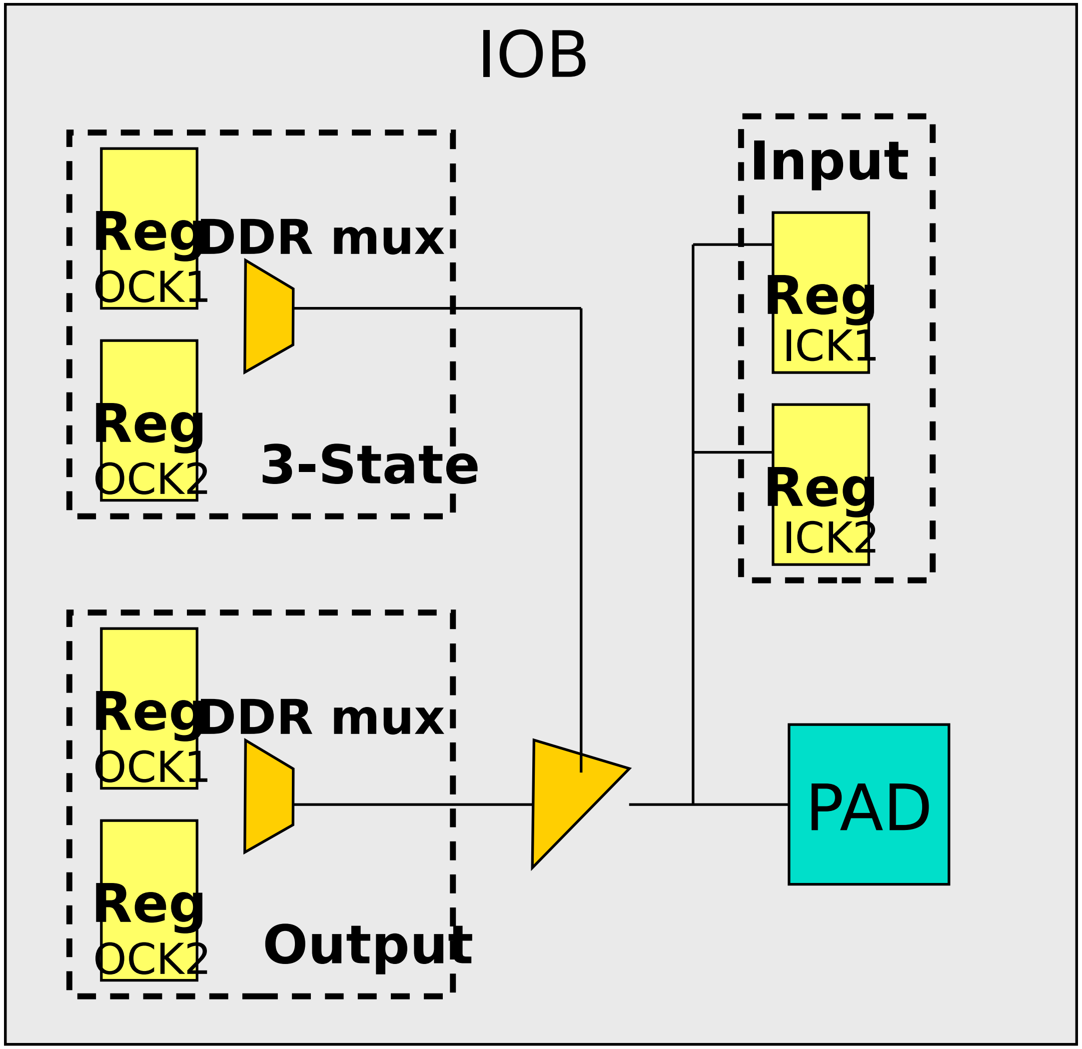




½ slice (top half), out of 46.080



IOB Element



IOB

- Input path
  - Two DDR registers
- Output path
  - Two DDR registers
  - Two 3-state DDR registers
- Separate clocks for I & O
- Set and reset signals are shared
  - Separated sync/async
  - Separated

Embedded 18 kb Block RAM

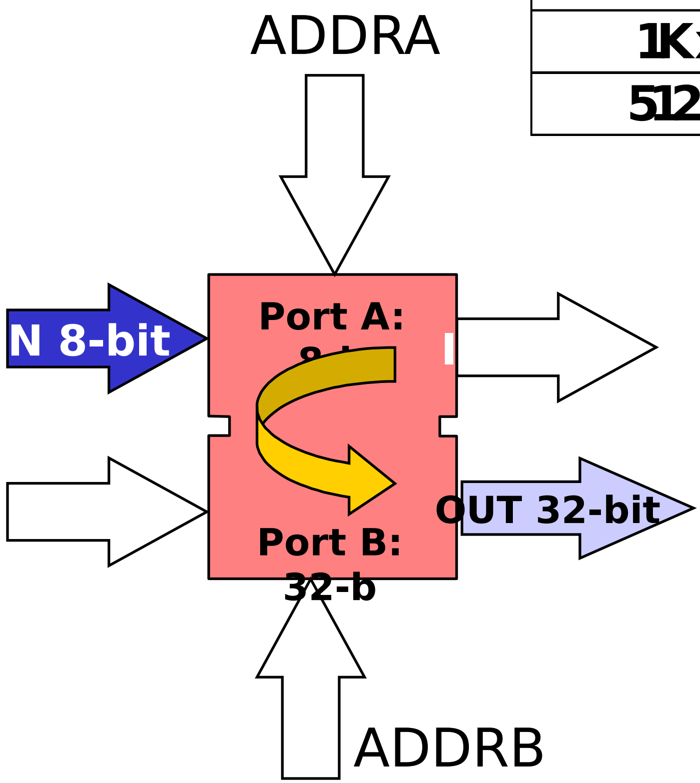
- Up to 3 Mb on-chip block RAM
- High internal buffering bandwidth
- Clocked write **and** read

	18Kbit block RAM
	Parity bit locations (parity in/out busses)
	Data width up to 36 bits
	3 WRITE modes
	Output latches Set/Reset
	True Dual-Port RAM
	Independent clock (async.) & control

True Dual-Port™ Configurations

Configurations available on each port:

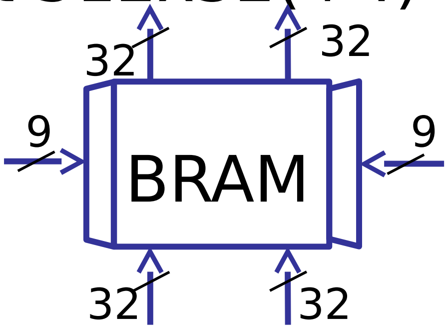
Configuration	Depth	Data bits	Parity bits
16Kx1	16Kb	1	0
8Kx2	8Kb	2	0
4Kx4	4Kb	4	0
2Kx9	2Kb	8	1
1Kx18	1Kb	16	2
512x36	512	32	4



- Independent port A and B configuration:
  - Support for data width conversion including parity bits

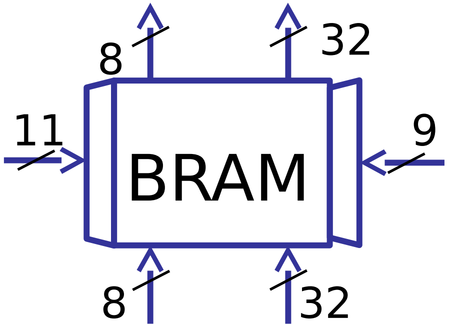
How to

Block RAM :
just instantiate template
2-port 512x32(+4)




```
RAMB16_S36_S36 inmem
    (// port A
     .CLKA(wb.clk), .SSRA(wb.rst),
     .ADDRA(bram_addr),
     .DIA(bram_data), .DIPA(4'h0),
     .ENA(bram_ce), .WEA(bram_we),
     .DOA(doa), .DOPA(),
     // port B
     .CLKB(wb.clk), .SSRB(wb.rst),
     .ADDRB({3'h0,rdc}),
     .DIB(32'h0), .DIPB(4'h0),
     .ENB(1'b1),.WEB(1'b0),
     .DOB(dob), .DOPB());
```

2048x8 512x32



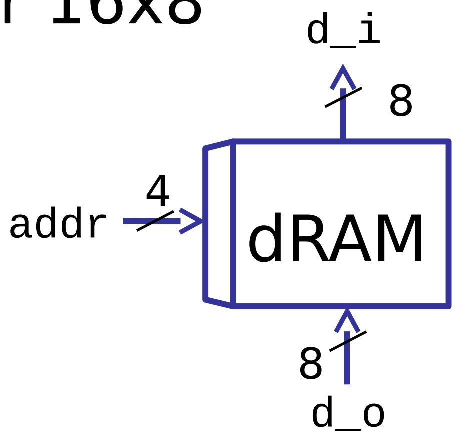

```
RAMB16_S9_S36 inmem
    (// port A
     …
     // port B
     … );
```

Distributed RAM

- Virtex-II LUT can implement:
  - 16 x 1-bit synchronous RAM
  - Synchronous write
  - Asynchronous read
    - D flip-flop in the same slice can register the output
- Allow fast embedded RAM of any width
  - Only limited by the number of slices in each device
  - Example: RAM 16 x 48-bit fits in 48 LUTs

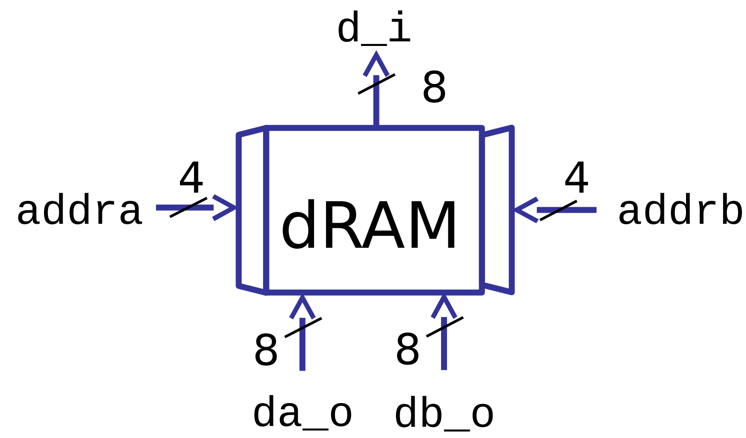
How to

Distributed RAM : 8 LUTs
1-adr 16x8




```
logic [7:0] mem0[0:15];

always_ff @(posedge clk)
     if (wr) begin
           mem0[addr] <= d_i;
     end

assign d_o=(rd) ? mem0[addr] : 8'h0;
```

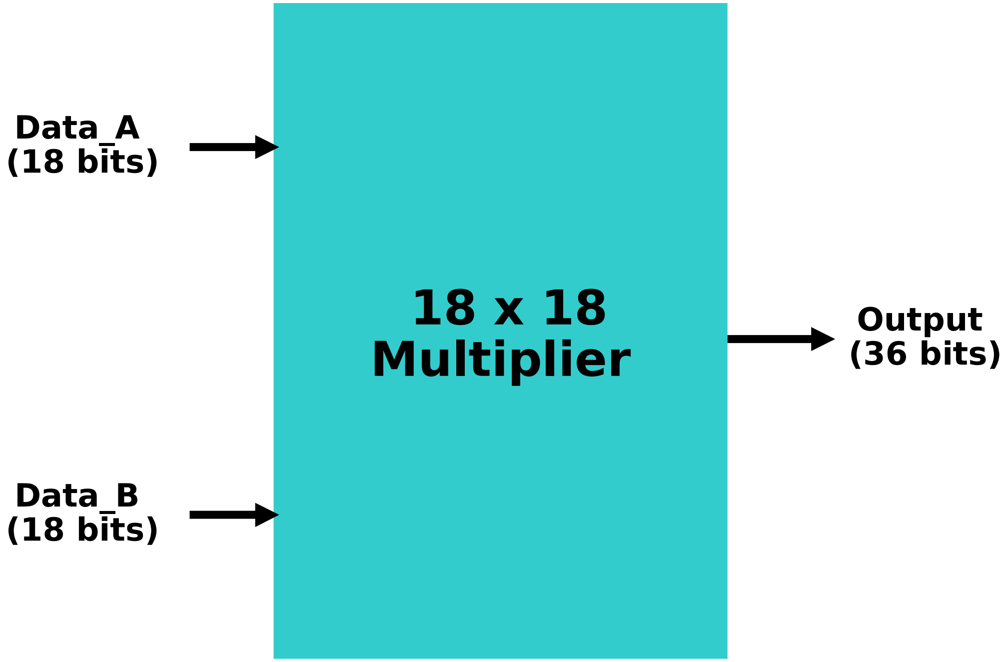
Distributed RAM : 16 LUTs
2-adr 16x8




```
logic [7:0] mem0[0:15];

always_ff @(posedge clk)
     if (wr) begin
           mem0[addra] <= d_i;
     end

assign db_o = (rdb) ? {mem0[addrb]
                    : 8'h0;
assign da_o = (rda) ? {mem0[addra]
                    : 8'h0;
```

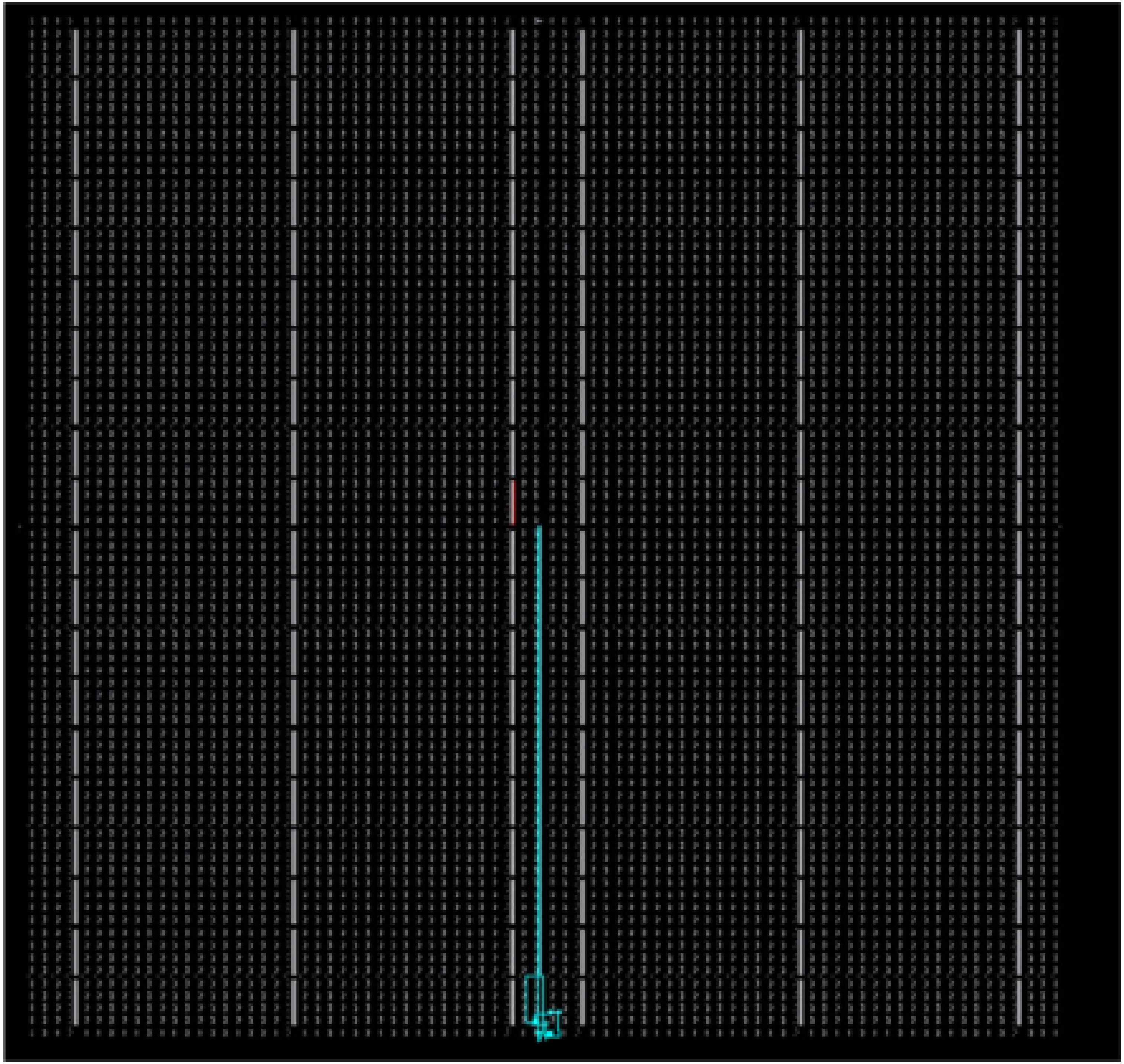
18 x 18 Multiplier

- Embedded 18-bit x 18-bit multiplier
  - 2's complement signed operation
- Multipliers are organized in columns



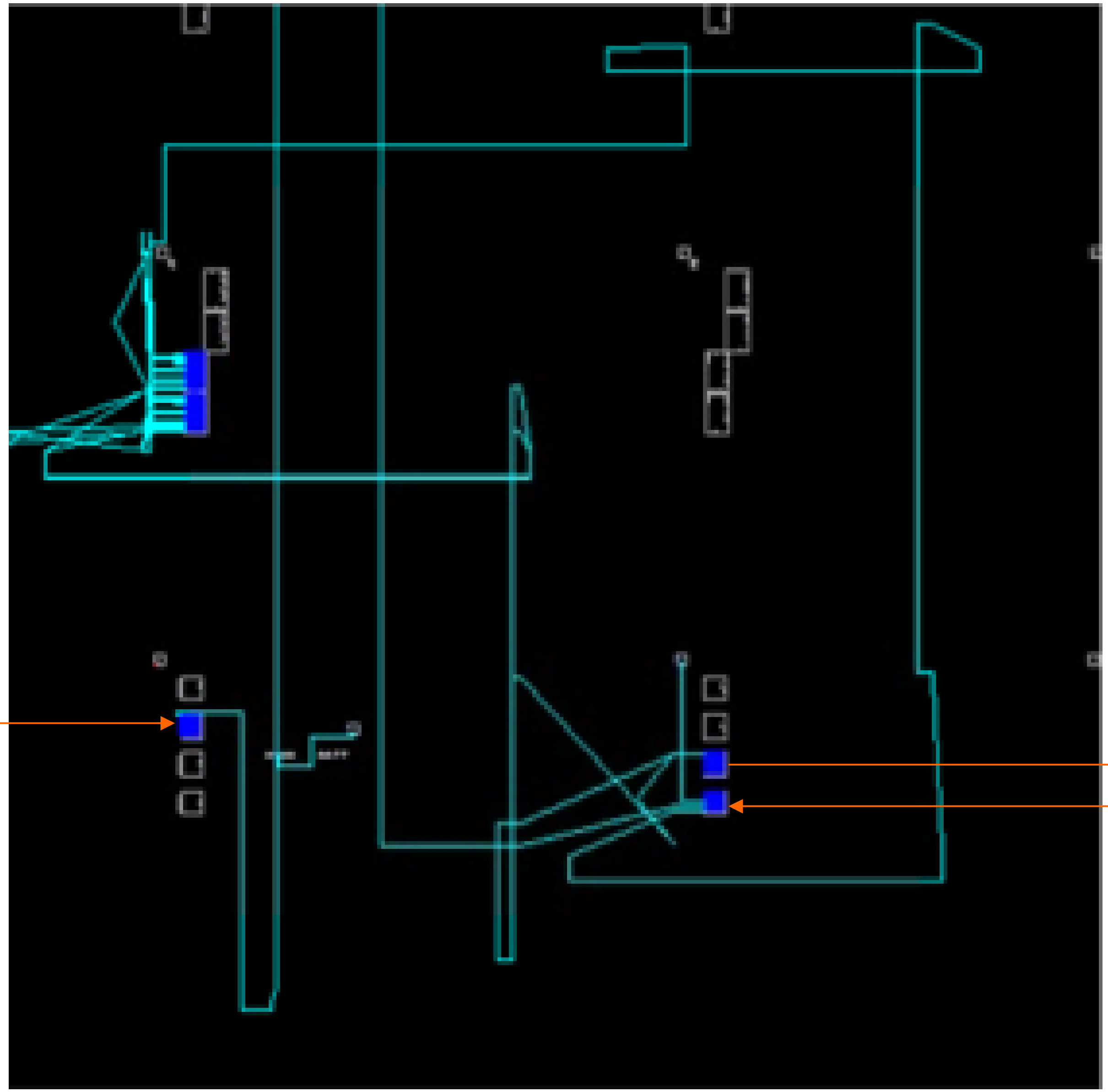
counter

```
module dec(
    input clk,rst
    output  u);

    reg u;
    reg [3:0] q;

    always_ff @(posedge clk or posedge rst)
   if (rst)
     q <= 4'h0;
   else if (q == 9)
     q <= 4'h0;
   else
     q <= q+1;

    always_ff @(posedge clk)
   if (q == 9)
     u <= 1'b1;
   else
     u <= 1'b0;
endmodule
```



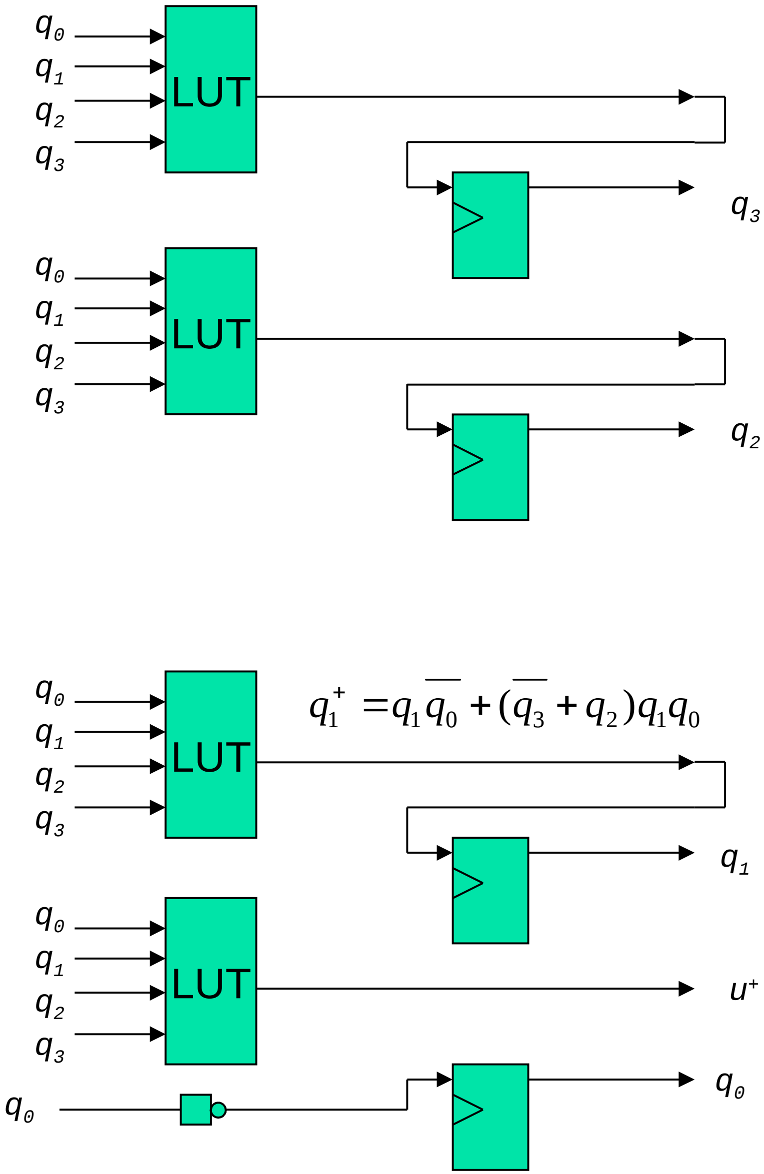
2 slices

clk

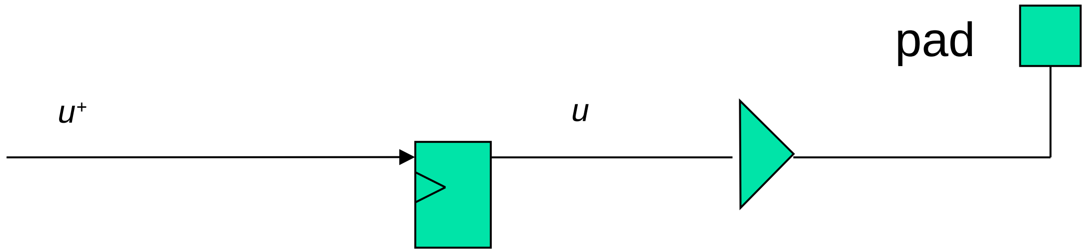


u

rst



2 slices = 4 LUTs



I/O-buffer