TSEA44: Computer hardware – a system on a chip

Lecture 5: Lab2 intro, Pitfalls when coding, debugging

LINKÖPING UNIVERSITY

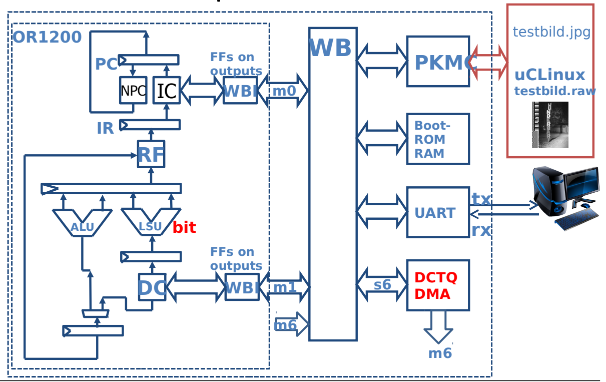
Agenda

- Lab2 introduction
- Pitfalls when writing code
- Debugging

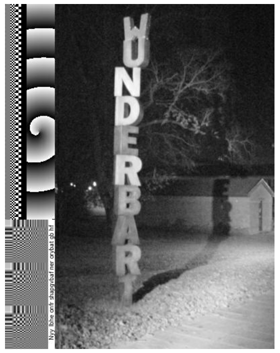
Lab 2 – A JPEG accelerator

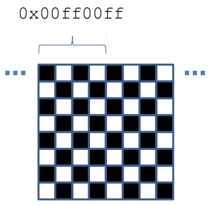
1. Design HW
2. Change existing software jpegfiles under uCLinux
   a) insert your accelerator
   b) insert your DMA
   c) insert your instruction

Our FPGA computer with accelerator



Raw image format in memory

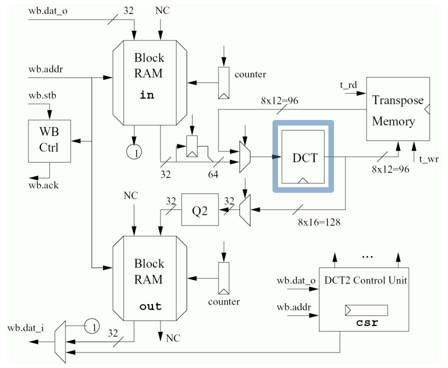




8 bit pixels [0,255]
4 pixels/word

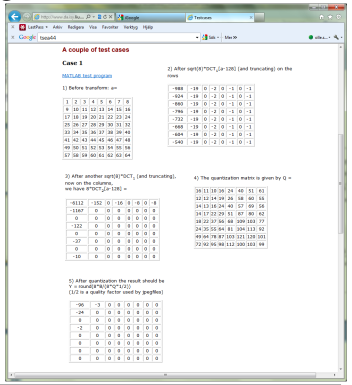
Somewhere 128 must be subtracted from each pixel!

Proposed architecture



Testcases available

- Lab page of the course webpages
- Includes code for quantization


http://www.da.isy.liu....
iGoogle
Testcases
LastPass Arkiv Redigera Visa Favoriter Verktyg Hjälp
Google tsea44
Sök Mer
olle.s...

A couple of test cases

Case 1

MATLAB test program

1) Before transform: a=

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

2) After sqrt(8)*DCT$_1$[a-128] (and truncating) on the rows

-988	-19	0	-2	0	-1	0	-1
-924	-19	0	-2	0	-1	0	-1
-860	-19	0	-2	0	-1	0	-1
-796	-19	0	-2	0	-1	0	-1
-732	-19	0	-2	0	-1	0	-1
-668	-19	0	-2	0	-1	0	-1
-604	-19	0	-2	0	-1	0	-1
-540	-19	0	-2	0	-1	0	-1

3) After another sqrt(8)*DCT$_1$ (and truncating), now on the columns, we have 8*DCT$_2$[a-128] =

-6112	-152	0	-16	0	-8	0	-8
-1167	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-122	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-37	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-10	0	0	0	0	0	0	0

4) The quantization matrix is given by Q =

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

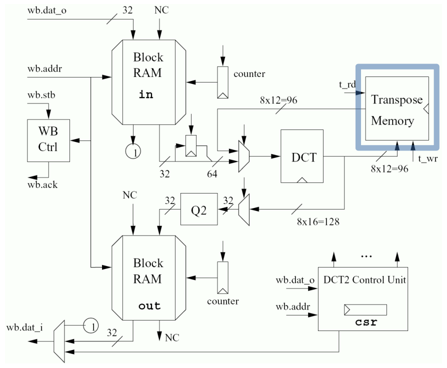
5) After quantization the result should be
Y = round(8*B/(8*Q*1/2))
(1/2 is a quality factor used by jpegfiles)

-96	-3	0	0	0	0	0	0
-24	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-2	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

DCT module

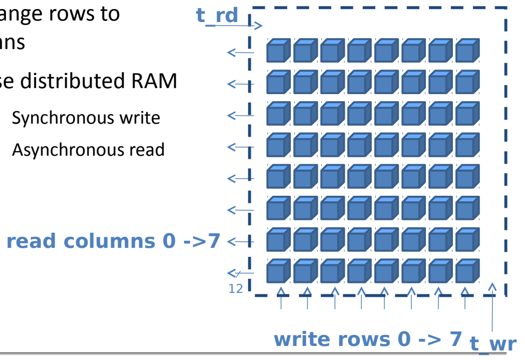
- Given to you
  - 1D DCT
    - 8 in ports (12 bits), 8 out ports (16 bits)
    - Fix point arithmetic
    - Straightforward implementation of Loeffler's algorithm

Proposed architecture

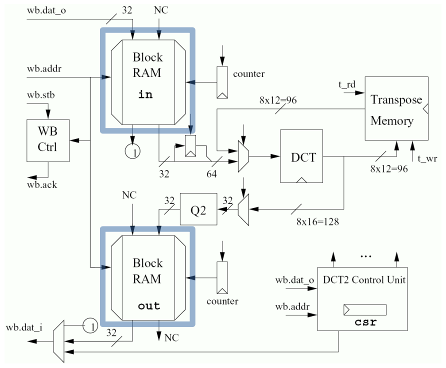


Transpose Memory

- Rearrange rows to columns
  - Use distributed RAM
    - Synchronous write
    - Asynchronous read



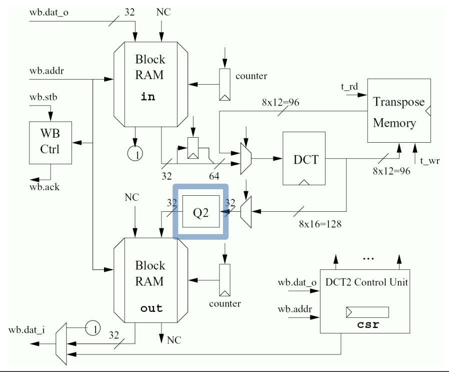
Proposed architecture



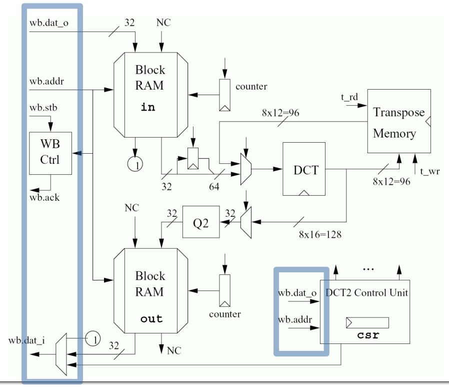
Block RAM

- Different timing
- ”Normal” SRAM
  - Asynchronous read
  - Asynchronous write
- Block RAM in Virtex 2
  - Synchronous read
  - Synchronous write

Proposed architecture

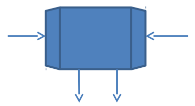


Proposed architecture

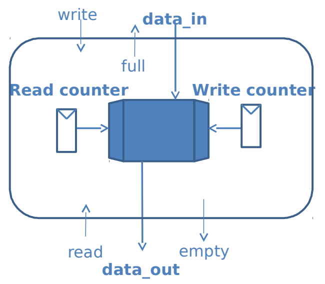


Some ideas

You can read 8 pixels per clock, If you use both ports

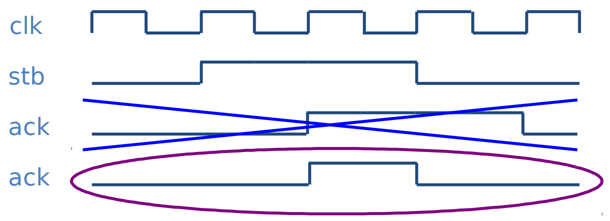


You can rebuild the BRAM to a FIFO



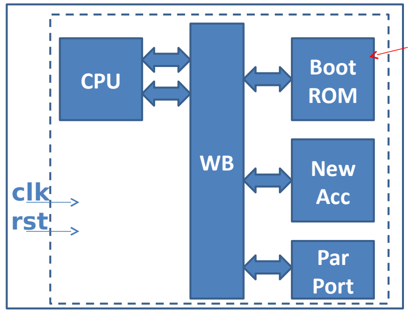
Some notes on the WB I/F

- Be careful with wb.ack



Test benches – 2 alternatives

1) Simulate the whole computer – make sim



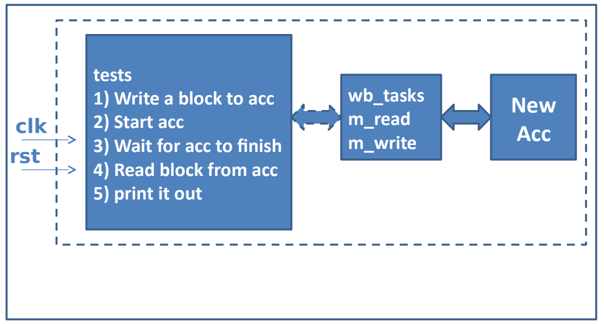
Insert some code In the beginning of the monitor **mon2.c**

There are some alternatives to uncomment

Tip: You can write to parport to make it easier to find things in ModelSim

Test benches – 2 alternatives

2) Simulate the accelerator – make sim_jpeg



wb_tasks.sv

```
module wishbone_tasks(wishbone.master wb);
   int result = 0;
   reg oldack;
   reg [31:0] olddat;

   always @(posedge wb.clk) begin
      oldack <= wb.ack;
      olddat <= wb.dat_i;
   end

   task m_read(input [31:0] adr, output logic [31:0] data);
      begin
      @(posedge wb.clk);
      wb.adr <= adr;
      wb.stb <= 1'b1;
      wb.we  <= 1'b0;
      wb.cyc <= 1'b1;
      wb.sel <= 4'hf;

      @(posedge wb.clk);
      #1;
      while (!oldack) begin
        @(posedge wb.clk);
              #1;
      end
```

```
      wb.stb <= 1'b0;
      wb.we  <= 1'b0;
      wb.cyc <= 1'b0;
      wb.sel <= 4'h0;

      data = olddat;
      end
   endtask // m_read
   ...
endmodule // wishbone_tasks
```

Potential pitfalls when creating a design

- What can go wrong?
  - Design mistakes
  - Synthesis errors
  - Runtime errors
- Crossing clock domains
  - Handshaking
  - Asynchronous FIFOs

A design bug

- Symptom: The boot sequence of uClinux hangs after a second when the Icache is on.
- Uclinux boots ok with Icache off
- No problems detected in the monitor when the icache is on

First try

- Modify the testbench so uClinux is present in SDRAM models
- Add interesting signals to the wave window
- Run the simulation over night

Oops...

- In the morning the simulation was not running any longer
- The log files had filled up all free space on the fileserver...
  - ... which promptly crashed, causing all sorts of merriment

Handling long simulation runtimes

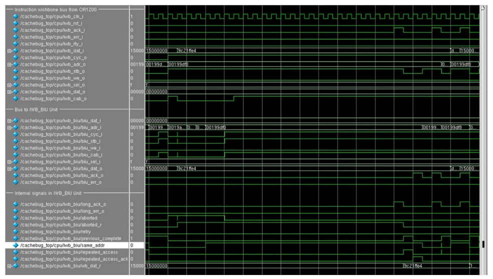
- Use checkpointing to reduce/eliminate the need for logging
  - Add no signals to wave window (and log for that matter)
  - Modify UART so printouts are displayed in the transcript window (using $display())
  - run 100 ms; checkpoint 100ms.chk
  - run 100 ms; checkpoint 200ms.chk
  - run 100 ms; checkpoint 300ms.chk
  - ...

Handling long simulation runtime, cont.

- Now you can pinpoint the time interval where the crash happened
  - Restore the checkpoint in Modelsim that occured closest before the actual crash
  - vsim -restore 600ms.chk
  - Debug as usual (by adding signals to wave window/etc)

So what was the bug?

- Cacheline filled up incorrectly (AAAA AAAA CCCC DDDD instead of AAAA BBBB CCCC DDDD)


Instruction wishbone bus from OR1200
/cachebug_top/cpu/iwb_clk_i
/cachebug_top/cpu/iwb_rst_i
/cachebug_top/cpu/iwb_ack_i
/cachebug_top/cpu/iwb_err_i
/cachebug_top/cpu/iwb_rty_i
/cachebug_top/cpu/iwb_dat_i
/cachebug_top/cpu/iwb_cyc_o
/cachebug_top/cpu/iwb_adr_o
/cachebug_top/cpu/iwb_stb_o
/cachebug_top/cpu/iwb_we_o
/cachebug_top/cpu/iwb_sel_o
/cachebug_top/cpu/iwb_dat_o
/cachebug_top/cpu/iwb_cab_o
Bus to IWB_BIU Unit
/cachebug_top/cpu/iwb_biu/biu_dat_i
/cachebug_top/cpu/iwb_biu/biu_adr_i
/cachebug_top/cpu/iwb_biu/biu_cyc_i
/cachebug_top/cpu/iwb_biu/biu_stb_i
/cachebug_top/cpu/iwb_biu/biu_we_i
/cachebug_top/cpu/iwb_biu/biu_cab_i
/cachebug_top/cpu/iwb_biu/biu_sel_i
/cachebug_top/cpu/iwb_biu/biu_dat_o
/cachebug_top/cpu/iwb_biu/biu_ack_o
/cachebug_top/cpu/iwb_biu/biu_err_o
Internal signals in IWB_BIU Unit
/cachebug_top/cpu/iwb_biu/long_ack_o
/cachebug_top/cpu/iwb_biu/long_err_o
/cachebug_top/cpu/iwb_biu/aborted
/cachebug_top/cpu/iwb_biu/aborted_r
/cachebug_top/cpu/iwb_biu/retry
/cachebug_top/cpu/iwb_biu/previous_complete
/cachebug_top/cpu/iwb_biu/same_addr
/cachebug_top/cpu/iwb_biu/repeated_access
/cachebug_top/cpu/iwb_biu/repeated_access_ack
/cachebug_top/cpu/iwb_biu/wb_dat_r

What if you cannot find a bug during simulation?

- Very likely you have some undefined behavior in your design
  - Race condition in RTL code (blocking vs non-blocking assignment)
  - Incorrect use of ”don't cares”
  - You are not crossing clock domains correctly
  - etc.
- Not so likely:
  - You have triggered a bug in the CAD tools

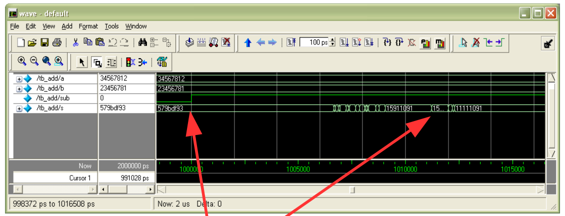
Clock domain crossing

- Why do we need synchronous designs?
  - Race conditions
  - Metastability
- Crossing clock domains
  - (Avoid if possible)
  - Using handshakes
  - Using asynchronous FIFOs
  - Your own solution
    - (Only if you like debugging systems where bugs cannot be deterministically reproduced...)
- Do not forget that the reset signal has to be passed to each clock domain!

Troubleshooting

Available for lab0!
make sim_lab0_sdf
See lab webpage

- Post Place-and-Route (PAR) simulation
  - Generate a new netlist using netgen
  - Simulation done with LUTs and FF



32-bit add/sub example:
Output `s` takes 15ns to stabilize after `sub 0->1`

LINKÖPING UNIVERSITY

Testbenches that work with PAR netlists

- Avoid violating setup and hold times of flipflops
  - Delay test values
- Test results at the end of the clock cycle
  - Test values at the clock cycle transition, before updates moved on from input flipflops

```
initial begin // Test adder
     @(posedge clk);
     #4; // delay after clockedge
     a <= 5;
     b <= 3;
     @(posedge clk);
     if (result != 8) begin
        $display("Adder fail");
        $stop;
    end
end
```

LINKÖPING UNIVERSITY

Simulation ok, but still not working?

- Add measurement logic to the FPGA Design
  - Use switches and LEDs
- Chipscope/Signaltap
  - Add logic analyzer function to the FPGA design
  - Store samples in blockRAM or similar
  - Communicate with PC over JTAG
- Warning!
  - Many people think signaltap/chipscope replace simulation. It does not! Better to spend time writing better testbench

LINKÖPING UNIVERSITY

www.liu.se

LINKÖPING UNIVERSITY