Prof. Dr.-Ing. Hofmann
Integrated Electronic Systems Lab
Merckstr. 25, D-64283 Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT

HDL Lab Manual 2012

Dipl.-Ing. Boris Traskov

August 5, 2012

Contents

1 Introduction

In this lab the design of a complex digital circuit, i.e. a THUMB processor with multiple pipeline stages, will be practised in teams of four students. System modelling will be performed in the hardware description language Verilog (optionally SystemVerilog). Students will practise design space exploration within a typical design cycle using a 130nm CMOS standard cell library at 1.2V. Project work will be guided, but self-organised. I.e. given a range of proposals and best practises, groups determine responsibilities and organisation themselves.

1.1 Goals

- Use Verilog to model a processor that is defined on instruction set level.
- Go through the digital design process encompassing specification, register-transfer-level modelling, simulation/verification, synthesis and gate-level simulation.
- Evaluate the system's performance and resolve bottlenecks.
- Practice group work, documentation and presentation techniques.
- Apply course knowledge and use industry-grade tools, particularly:
  - Modelsim 6.5 (Mentor Graphics)
  - Design Compiler (Synopsys)
  - bash, TCL, make
  - GNU C toolchain for ARM

1.2 Expected Outcome

This lab serves as preparation for a B.Sc./M.Sc. thesis in digital design at the Integrated Electronic Systems Lab. The practised methodology can directly be used as a template for a thesis and extended towards physical implementation targeting ASIC or FPGA.

2 Primary Objectives

2.1 Processor Design in Verilog

Design a processor that can execute *Thumb* instructions for the instruction set given in the Thumb Quick Reference Card [5]. Implement all instructions in the following sections: Move, Ad, Subtract, Multiply, Compare, Logical, Shift/Rotate, Load, Store, Push, Pop, If-Then, Branch, Extend, Reverse, No Op. Do not implement Processor State Change and Hint. Refer to the ARM Architecture Manual [1] for a detailed description of the instruction set.

The entire program code is stored in a 4Kx16 fully-buffered, single-port random access memory (RAM). 4Kx16 means that it has 4096 entries (depth) of 16 bit (width) each and it can store 8KB (B = byte, b = bit) in total. Fully-buffered means that its inputs as well as its outputs are registered. On a write the memory contents update with the next rising edge of the clk input. On a read the data is delayed by one cycle before appearing at the ouptut. This memory can be clocked at up to 523 Mhz.

Internal Structure

An instruction set does not yet define the implementation of the cpu. It merely defines what registers exist, what instructions the cpu can process, what an instruction means etc. Therefore the architecture is up to you, with only one constraint: Use at least 2 pipeline stages. Hint: Try to stick to the basic structure in the section “Best Practices”.

2.2 Register-Transfer-Level Simulation and Verification

There are many ways to verify a design’s correctness. In this lab you will create a *directed test bench* and simulate it in Modelsim (a tutorial is provided in [4]). This means you will define test cases in the form of C-programs as in appendix C.1. Using the provided makefile in appendix C.2 you can compile your test programs into ARM-assembler language files (.asm), executable and linkable files (.elf) and binary files (.bin).
Keep these programs simple as you will need to trace the program execution on your waveform viewer. When writing test programs also specify the correct expected results (golden model).
Once this is done you will simulate your DUT (device under test, i.e. cpu) with these test cases as memory content (stimulus) and compare the final memory content with the expected results.

- Review and understand the provided testbench and memory in the appendix. Use it as your starting point and adapt it where needed.
- Write test programs (and document them) to ensure that all instructions work properly.
- You are allowed to share test programs and verify your design with other groups’ test programs. Give them credit in your report!
- Include an active-high finish_out signal in your design and assert it, when your program is done.

2.3 Synthesis and Design Evaluation

During the design process you will frequently synthesize your design with Synopsys Design Compiler using a 130nm standard cell library and Design Ware Building Blocks (DWBB). This will give you key metrics that will guide your design process. In particular, the synthesis tool will tell you how fast your circuit will be.
As a rule of thumb: Code that you write should be synthesized by the end of the day. (In the beginning try lots of small iterations. Later - when you know what you are doing - try longer iterations.) Appendix D is a tutorial on synthesis. It gives you a good starting point and produces all reports you need. Refer to [2] for in-depth documentation.
Take a look at your synthesis reports and understand key metrics:

- frequency
- (critical path delay)
- area
- power

Be able to explain these metrics. When optimizing your design observe how key metrics correlate with each other.

2.4 Gate-Level Simulation

Synthesis transforms your register-transfer-level design (RTL) into a gate-level design (GL). This is a net list of standard cells with their corresponding delays annotated. Simulating on this net list allows you to verify functionality as well as timing. A tutorial is provided in appendix E.
Once at the end of each week make sure, that you your example programs simulate correctly as in your RTL simulations. For fully-synchronous single-clocked designs you should not expect any surprises here. In asynchronous or multi-clock designs you'll be able to find bugs on gate-level that did not occur at register-transfer-level.
When you simulate on GL look at the waveforms and observe the switching activity. Review the terms slack, critical path and setup time.

3 Secondary Objectives

When you are confident to have a functionally correct and synthesizable design you are ready to advance to the secondary objectives. The main task in this section is to **optimize for execution time** by using the techniques described in the following subsections. The basic optimization cycle starts with an evaluation of your current design. Once you have found and understood a performance bottleneck you will think of ways to push the limits, implement additional circuitry and start over. Document your key figures and reasoning in each design refinement step. In most cases there is no optimal solution (or it's hard to tell before you try it out). Before you start coding any of the following extensions:

- Write a program, to demonstrate the extension's effectiveness. Calculate the speed increase. Note that sometimes it suffices to recompile an existing program with different compiler optimizations, i.e. -O3 instead of -O0 in the makefile)
- Think of how you want to integrate it in your processor. Is it an additional pipeline stage? Is it a modification to one (or more) existing pipeline stage(s)?
- Always KISS: "Keep it Simple and Stupid!" (If nothing else, remember this)

After (and while) designing an extension:

- Make sure that you do not break your other test programs.

3.1 Instruction/Data Chache

If memory bandwidth lags behind cpu performance:

- Build a configurable cache with parameters:
  - CACHE_ON=[0 | 1]
    determines if a cache is instantiated (1) or not (0).
  - CACHE_SIZE= [2 | 4 | 8 | 16 | 32]
    determines the number of words in your cache.
  - Additional parameters as needed (depending on policy).
- Cache policy and associativity is up to you. (KISS)

3.2 Branch Prediction

If a long pipeline causes long latency at branches:

- Build a branch prediction unit with parameters:
  - BP_ON=[0 | 1]
    determines if a branch predictor is instantiated (1) or not (0).
  - Additional parameters as needed (depending on policy).
- Algorithm is up to you. (KISS)

3.3 Superscalar Execution

If cpu performance lags behind memory bandwidth:

- Duplicate (triplicate, ...) the execution unit and add additional circuitry for decoding and distributing instructions and handling hazards.
- Use parameters:
  - SS_<UNIT>=[(default: 1) | 2 | 3...]
    indicating which unit is duplicated, triplicated...
    Replace <UNIT> with DEC, EX or other appropriate name
  - Additional parameters as needed.

3.4 Balanced Pipeline

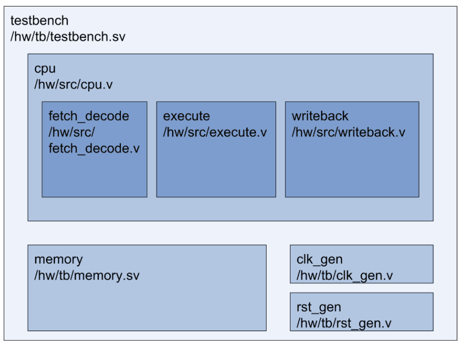
If cpu performance lags behind memory bandwidth:

- Split execute stage in two or more stages and add additional circuitry to handle hazards.
- If you have registered all your execute-stage outputs properly, this is an easy task. Look up the DesignCompiler command “balance_registers”.

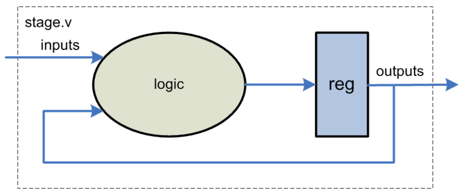
4 Best Practices

4.1 Coding

- Draw your schematics/state machines on paper before coding!
  - Do not try to save paper!
  - Clearly mark combinational logic in one color and non-combinational in another. (please not red/green)
  - leave out clocks and resets from your drawing... too messy
- Make one separate file for each module. Recommended granularity: A stage is a module, e.g. "execute.v, decode.v, ..." This split is actually finer-grained than absolutely necessary, but gives a nice, clean design.
  (Good to know: By default the synthesis tool performs logic optimizations only within a module. Thus it can ”divide and conquer” the synthesis problem. The drawback is that it will not try to move logic from one module to another. But then again, you can explicitly tell the synthesis tool to do cross-module optimizations. For large designs this often leads to out-of-memory situations.)
- Instantiate all synthesizable modules within "cpu.v". Then instantiate "cpu.v" in ”testbench.sv". This gives a nice, clean structure.



- Register all outputs of each stage, i.e. the output is driven by a register directly!



This ”latched mealy” machine is a very fast and safe technique to model pipelined data paths. (Mealy and Moore are taught in classes because they are difficult enough to confuse students. They are rarely used though, because of some poor properties: When connected together Mealy machines produce long combinational paths. Moore does not yield such long combinational paths but the basic drawback remains.)

- Decide upon a naming convention for signals, registers, modules, constants,...! Consistency helps when your code base grows larger and larger.
- Use of SystemVerilog syntax (easier, less confusing and still synthesizable) is recommended (not mandatory).
  - logic instead of wire/reg
  - always_ff/always_comb instead of always
  - file names end in .sv instead of .v
- Do not reinvent the wheel: Use the built-in operators "+" and "-" for addition and subtraction. The synthesis tool will infer the right DWBB architecture for you.
- No tri-state buses and no "inout" ports
- Port connections only by name, not by position
- Tasks/functions only if absolutely necessary
- Save your files often (at least once before lunch break and before leaving in the evening)! You can use a revision control system if you like.
- Comments, comments, comments

4.2 Tools

- Do not run any tool in the same working directory in two shells at the same time. Instead have seperate working directories for each instance of the respective tool.
- Use the graphical user interface (GUI) to get familiar with the functionality. Later, you can observe the commands issued by the GUI to create your own scripts. The latter is preferred because it guarantees reproducibility. Also many useful commands/options are not available in the GUI.

4.3 Group Work

- Work together, discuss regularly and often.
- At least once a day (in the morning), discuss what each of you will do today.
- Work in pairs of 2 for a while, then come together to integrate your work. Explain what you did to the others. Make only small iterations!
- Make only small iterations! REALLY!
- Inform each other of your schedules/absences well in advance.

5 Deliverables

Make a folder tree as indicated below and place your files and your final report named

hdllab2012_xx.pdf

in the corresponding sub-folder. Pack everything in a zip-file named

hdllab2012_xx.zip

(xx is your group number) and email it to

boris.traskov@ies.tu-darmstadt.de
by
Friday evening, August 24.

1. Folder structure
   - /**doc** (put your report here)
   - /**hw**/**src** (RTL Verilog source code)
   - /**hw**/**tb** (testbench source code)
   - /**hw**/**scripts** (compilation, simulation, synthesis scripts)
   - /**hw**/**rep** (synthesis reports: area, power, timing, references)
   - /**hw**/**mapped** (gate-level .ddc and .v netlists)
   - /**hw**/**unmapped** (gtech .ddc)
   - /**sw**/**src** (makefile, c-source files and binaries)
2. Report (12 pages, English or German)
   - General outline
     - Introduction / motivation / goals / work distribution (1 page)
     - Implementation / technical work (10 pages)
       - Design
       - Verification
       - Synthesis Results (summarized, only key figures + discussion)
       - (other, e.g. comparison between different architectures... you can go crazy here)
     - Evaluation / conclusion (1 page)
     - Appendix
       - Anything you deem fit.
   - Hints
     - There is no cookbook recipe (and only few mathematical formulas) for processor design, only good/bad experience. Therefore reports and articles in this domain need to focus on the reasoning, i.e. why a particular design was chosen. Sections on design iterations, evaluation strategies, comparisons between different options/-configurations and so on are most interesting to read.
     - Use passive mode: ”The registers are compromised of d-flipflops” instead of ”we used d-flipflops for the registers”. In the work distribution (beginning) and conclusion (very end) sections you may use active mode, i.e. ”Boris was responsible for the decode stage” or ”the authors will continue verification after tape-out”.

  - Express yourself clearly and in a concise form.
  - This is a good occasion to learn LaTeX, because you will need it for your thesis. After the lab you are given a full week to write your report for exactly this reason.
- Sad but true
  - Plagiarism is embarrassing! Point out your own as well as others' original work. Cite when you use other people's results (even when paraphrasing). Ask if in doubt. All reports and source code at IES are checked with automated tools.

6 Examination

The examination is mandatory, oral and takes place in groups of 4. It lasts roughly 30-45 minutes and the date is by agreement, preferably in the week after report submission. Use the following collection of questions for preparation. Read "explain" as "tell a non-technical person how it works." If you are able to do so, then you can also explain it to your examinor.

1. Know your group's design
   - Explain your general architecture. (What where your considerations when chosing this particular one?)
   - Explain some line of your entire source code.
   - Explain something that you have written (or omitted) in your report.
   - Explain how an instruction passes through your design.
   - Which is the most expensive operation in terms of time/area/power.
2. Verilog
   - Write code for: DFF, sensitive to rising/falling edge with/without enable, with (a)synchronous reset
   - Write code for: purely combinational barrel shifter that shifts din 0, 1, 2 or 3 bits right and drives dout with the result. Empty bits are filled with zeros. selection is based on signal "shift" (00, 01, 10, 11). din/dout have a width of 8.
   - Write code for: some other simple logic/FSM ...
3. Digital Design
   - Explain: Moore, Mealy, latched-Mealy
   - Explain: 2's Complement, signed and unsigned representaion, carry and overflow
   - Explain: combinational and non-combinational
   - Explain: D-FF and Latch.
   - Explain: setup time and hold time
   - Explain: gtech library, synthetic library, Design Ware Building Blocks library
   - Explain: critical path
   - Explain: maximum frequency
   - Explain: the power report
   - Explain: the timing report
   - Explain: the area report
   - Explain: the reference report
4. Pipelining
   - What is the idea behind pipelining?
   - Explain: hazard
   - What types of hazards do you know? When do they occur? How can you resolve hazards?

- What is a balanced pipeline?
- Quantify the speed increase in a 3-stage (N-stage) pipelined processor compared to a single-stage processor (best case, worst case). How does setup time of the registers affect the ideal model (no setup time)? What effects influence the speed increase?

5. Other questions related to anything you did in the lab.

A IES Servers and Tools

A.1 Login Procedure

```
################################################################
### IES SERVERS AND TOOLS ###
################################################################

#start the lab pc
#enter username + password
#open a terminal
#login to an IES server and work there (use 64bit machines for gcc)
#64bit-server := [ falbala | adonix ]
#32bit-server := [ obelix | talentix | gibtermine ]
ssh -X username@server

#list available tools
module avail

#load tools
module load modelsim
module load syn/2010.12-SP1
module load hdllab

#verify loaded modules
module list

#start Design Compiler in graphical mode from dc_work directory
design_vision &

#start Modelsim from ms_work directory
design_vision &

#to unload a tool
module unload <module name>

#if a prgram crashes, find its process id (pid) with
ps
#thenn kill it with
kill <pid>
#you will need to do this sometimes with DesignCompiler!
```

tutorials/tools.txt

A.2 Design Compiler Configuration

```
set DC_LIB_DIR    "/cad/umc/UMC13"
set DC_LIB_NAME   "UMC CMOS 0.13u, version 2003"

set search_path   [concat $search_path                              \
                        $DC_LIB_DIR/UMCE13H210D3_1.2/design_compiler \
                        $DC_LIB_DIR/UMCE13H300D3_1.0/design_compiler \
                        $DC_LIB_DIR/UMCE13H350D3_1.0/design_compiler \
                        ../src]

set target_library      "umce13h210t3_tc_120V_25C.db
    umce13h300t3_tc_120V_25C.db umce13h350t3_tc_120V_25C.db"
set symbol_library      "umce13h210t3.sdb umce13h300t3.sdb umce13h350t3.sdb
    "
set synthetic_library   "dw_foundation.sldb standard.sldb"

set link_library    [concat * \
                            $target_library \
                            $synthetic_library \
]


set designer "Dipl.-Ing. Boris Traskov"
set hdl_keep_license FALSE
set_host_options -max_cores 16

# Define aliases
alias h history
alias rc report_constraint -all_violators
```

../../project/hw/dc_work/.synopsys_dc.setup

Note: this file must be either in your home directory or in the directory from which you invoke design_vision!

A.3 NxClient Configuration

Server
host: falbala.ies.e-technik.tu-darmstadt.de
port: 22

Desktop
Unix - Gnome
If working from inside the TU Darmstadt-network this works fine. From the outside you first need to VPN to the HRZ to get a TUD IP address. A simple description for this is available here:`http://www.hrz.tu-darmstadt.de/dienste/netz_und_internet/vpn_wlan/vpn_service/vpn_anleitungen/vpn_anleitung_idx.de.jsp`

B Verilog and SystemVerilog Templates

```
// Integrated Electronic Systems Lab
// TU Darmstadt
// Author:  Dipl.-Ing. Boris Traskov
// Email:   boris.traskov@ies.tu-darmstadt.de

`timescale 1 ns / 1 ps

 module cpu (clk, rst, addr);
   // PARAMETERS
   parameter ADDR_WIDTH = 13;

   // INPUTS
   input clk;
   input rst;

   // OUTPUTS
   output [ADDR_WIDTH-1:0] addr;

   reg  [ADDR_WIDTH-1:0] pc_Q;
   wire [ADDR_WIDTH-1:0] pc_D;

   wire [ADDR_WIDTH-1:0] addr;

   // DFF, synch reset
   always @(posedge clk)
   begin
     if (rst == 1'b1)
       pc_Q <= 0;
     else
       pc_Q <= pc_D;
   end

   assign pc_D = pc_Q + 1;
   assign addr = pc_Q;

 endmodule
```

../../project/hw/src/cpu.v

```
// Integrated Electronic Systems Lab
// TU Darmstadt
// Author:  Dipl.-Ing. Boris Traskov
// Email:   boris.traskov@ies.tu-darmstadt.de

`timescale 1 ns / 1 ps

module memory(
    clk,
    en,
    rd_en,
    wr_en,
    addr,
    din,
    dout
);

//stores this many halfwords (1halfword=16bit=2Byte)
parameter   MEM_DEPTH   = 2**12;

//addresses this many Bytes (1Byte = 8bit)
localparam  ADDR_WIDTH  = $clog2(MEM_DEPTH*2);

// PORTS
input   logic                           clk;
input   logic                           en;
input   logic                           rd_en;
input   logic  [0:1]                    wr_en;
input   logic  [0:1][7:0]               din;
output  logic  [0:1][7:0]               dout;
input   logic       [ADDR_WIDTH-1:0]    addr;

// MEM ARRAY AND INTERNAL SIGNALS
logic  [0:1][7:0]  ram [0:MEM_DEPTH-1];
logic  [0:1][7:0]  wr_halfword;
integer wr_i;

// WR_EN DECODER
always_comb begin
    wr_halfword = ram[addr[ADDR_WIDTH-1:1]];
    for(wr_i=0; wr_i<2; wr_i=wr_i+1) begin
        if (wr_en[wr_i]==1'b1) begin
            wr_halfword[wr_i] = din[wr_i];
        end
    end
end

// REGISTERED WRITE
always_ff@(posedge clk) begin
    if (en==1'b1) begin
        ram[addr[ADDR_WIDTH-1:1]] <= wr_halfword;
    end
end
```

```

// REGISTERED READ
always_ff@(posedge clk) begin
    if (en==1'b1) begin
        if (rd_en==1'b1) begin
            dout <= ram[addr[ADDR_WIDTH-1:1]];
        end
    end
end

endmodule
```

../../project/hw/tb/memory.sv

```
// Integrated Electronic Systems Lab
// TU Darmstadt
// Author:  Dipl.-Ing. Boris Traskov
// Email:   boris.traskov@ies.tu-darmstadt.de

`timescale 1 ns / 1 ps

module testbench();

// PARAMETERS
parameter MEM_DEPTH          = 2**13;     //8192 Bytes
parameter ADDR_WIDTH         = $clog2(MEM_DEPTH);
parameter string filename    = "../../sw/src/count32.bin";

// INTERNAL SIGNALS
integer file, status; // needed for file-io
logic            clk;
logic            rst;
logic            en;
logic            rd_en;
logic [ 1:0]     wr_en;
logic [15:0]     data_cpu2mem;
logic [15:0]     data_mem2cpu;
logic [ADDR_WIDTH-1:0] addr;

assign en        = 1'b1;
assign rd_en     = 1'b1;
assign wr_en     = 2'b0;

// CPU INSTANTIATION
cpu
cpu_i (
    .clk     (clk),
    .rst     (rst),
    .addr    (addr)
    // add more signals here
);

// MODULE INSTANTIATION
memory #(
    .MEM_DEPTH (MEM_DEPTH))
memory_i (
    .clk     (clk),
    .addr    (addr),
    .en      (en),
    .rd_en   (rd_en),
    .wr_en   (wr_en),
    .din (data_cpu2mem),
    .dout(data_mem2cpu));

//CLOCK GENERATOR
initial begin
    clk = 1'b0;
    forever #1  clk = !clk;
end

//RESET GENERATOR
initial begin
    rst         = 1'b0;
    file        = $fopen(filename, "r");
    #3 rst      = 1'b1;      // 3   ns
    status      = $fread(memory_i.ram, file);
    #2.1 rst    = 1'b0;  //2.1 ns
end

endmodule;
```

../../project/hw/tb/testbench.sv

C Software Templates

C.1 Test Programs

```c
// Integrated Electronic Systems Lab
// TU Darmstadt
// Author:  Dipl.-Ing. Boris Traskov
// Email:   boris.traskov@ies.tu-darmstadt.de
// Purpose: counts "i" from 0 up to 31

#define N 32

volatile int i = 0x76543210;    //marker in .bss field

main() {
    for (i=0; i<N; i++) {
    }
    return 0;
}
```

../../project/sw/src/count32.c

```

count32.elf:     file format elf32-littlearm
count32.elf
architecture: arm, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000

Program Header:
    LOAD off    0x00008000 vaddr 0x00000000 paddr 0x00000000 align 2**15
         filesz 0x00000204 memsz 0x00000204 flags rwx
private flags = 200: [APCS-32] [FPA float format] [software FP]

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         0000002c  00000000  00000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000004  00000200  00000200  00008200  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .comment      00000012  00000000  00000000  00008204  2**0
                  CONTENTS, READONLY
SYMBOL TABLE:
00000000 l    d  .text  00000000 .text
00000200 l    d  .data  00000000 .data
00000000 l    d  .comment   00000000 .comment
00000000 l    d  *ABS*  00000000 .shstrtab
00000000 l    d  *ABS*  00000000 .symtab
00000000 l    d  *ABS*  00000000 .strtab
00000000 l    df *ABS*  00000000 count32.c
00000200 g     O .data  00000004 i
00000000 g     F .text  0000002c main


```

```
Contents of section .text:
 0000 80b502af 084a0023 136004e0 064b1b68  .....J.#.`...K.h
 0010 5a1c054b 1a60044b 1b681f2b f6dd0023  Z..K.`.K.h.+...#
 0020 181cbd46 82b080bd 00020000           ...F........
Contents of section .data:
 0200 10325476                             .2Tv
Contents of section .comment:
 0000 00474343 3a202847 4e552920 342e312e  .GCC: (GNU) 4.1.
 0010 3100                                 1.
Disassembly of section .text:
00000000 <main> b580        push    {r7, lr}
00000002 <main+0x2> af02        add r7, sp, #8
00000004 <main+0x4> 4a08        ldr r2, [pc, #32]   (00000028 <.text+0x28>)
00000006 <main+0x6> 2300        movs    r3, #0
00000008 <main+0x8> 6013        str r3, [r2, #0]
0000000a <main+0xa> e004        b.n 00000016 <main+0x16>
0000000c <main+0xc> 4b06        ldr r3, [pc, #24]   (00000028 <.text+0x28>)
0000000e <main+0xe> 681b        ldr r3, [r3, #0]
00000010 <main+0x10> 1c5a       adds    r2, r3, #1
00000012 <main+0x12> 4b05       ldr r3, [pc, #20]   (00000028 <.text+0x28>)
00000014 <main+0x14> 601a       str r2, [r3, #0]
00000016 <main+0x16> 4b04       ldr r3, [pc, #16]   (00000028 <.text+0x28>)
00000018 <main+0x18> 681b       ldr r3, [r3, #0]
0000001a <main+0x1a> 2b1f       cmp r3, #31
0000001c <main+0x1c> ddf6       ble.n   0000000c <main+0xc>
0000001e <main+0x1e> 2300       movs    r3, #0
00000020 <main+0x20> 1c18       adds    r0, r3, #0
00000022 <main+0x22> 46bd       mov sp, r7
00000024 <main+0x24> b082       sub sp, #8
00000026 <main+0x26> bd80       pop {r7, pc}
00000028 <.text+0x28> 0200          lsls    r0, r0, #8
0000002a <.text+0x2a> 0000          lsls    r0, r0, #0
```

../../project/sw/src/count32.dasm

```
0000000: 80b5 02af 084a 0023 1360 04e0 064b 1b68  .....J.#.`...K.h
0000010: 5a1c 054b 1a60 044b 1b68 1f2b f6dd 0023  Z..K.`.K.h.+...#
0000020: 181c bd46 82b0 80bd 0002 0000 0000 0000  ...F............
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000200: 1032 5476                                .2Tv
```

../../project/sw/src/count32.bintxt

```
0000000: 80b5 02af 084a 0023 1360 04e0 064b 1b68  .....J.#.‘...K.h
0000010: 5a1c 054b 1a60 044b 1b68 1f2b f6dd 0023  Z..K.‘.K.h.+...#
0000020: 181c bd46 82b0 80bd 0002 0000 0000 0000  ...F............
0000030: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000040: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000050: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000060: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000070: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000080: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000090: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00000f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000100: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000110: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000120: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000130: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000140: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000150: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000160: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000170: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000180: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000190: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001a0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001b0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001c0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
0000200: 1f00 0000                                ....
```

../../project/sw/src/count32.goldtxt

```
// Integrated Electronic Systems Lab
// TU Darmstadt
// Author:  Dipl.-Ing. Boris Traskov
// Email:   boris.traskov@ies.tu-darmstadt.de)
// Purpose: copies "msg" to "dst"

#define N 46

static      char msg[N] = "A long time ago, in a galaxy far, far away...";
volatile    char dst[N];

main() {
    int i;
    for (i=0; i<N; i++) {
        dst[i] = msg[i];
    }
    return 0;
}
```

../../project/sw/src/memcpy46.c

```

memcpy46.elf:     file format elf32-littlearm
memcpy46.elf
architecture: arm, flags 0x00000112:
EXEC_P, HAS_SYMS, D_PAGED
start address 0x00000000

Program Header:
    LOAD off    0x00008000 vaddr 0x00000000 paddr 0x00000000 align 2**15
         filesz 0x00000230 memsz 0x0000025e flags rwx
private flags = 200: [APCS-32] [FPA float format] [software FP]

Sections:
Idx Name          Size      VMA       LMA       File off  Algn
  0 .text         00000044  00000000  00000000  00008000  2**2
                  CONTENTS, ALLOC, LOAD, READONLY, CODE
  1 .data         00000030  00000200  00000200  00008200  2**2
                  CONTENTS, ALLOC, LOAD, DATA
  2 .bss          0000002e  00000230  00000230  00008230  2**0
                  ALLOC
  3 .comment      00000012  00000000  00000000  00008230  2**0
                  CONTENTS, READONLY
SYMBOL TABLE:
00000000 l    d  .text   00000000 .text
00000200 l    d  .data   00000000 .data
00000230 l    d  .bss    00000000 .bss
00000000 l    d  .comment    00000000 .comment
00000000 l    d  *ABS*   00000000 .shstrtab
00000000 l    d  *ABS*   00000000 .symtab
00000000 l    d  *ABS*   00000000 .strtab
00000000 l    df *ABS*   00000000 memcpy46.c
00000200 l     O .data   0000002e msg
00000230 g     O .bss    0000002e dst
```

```
00000000 g     F .text  00000044 main


Contents of section .text:
 0000 80b581b0 00af3a1c 00231360 0ce03b1c  ......:..#.`..;.
 0010 19683b1c 1a68094b 9b5c094a 53543a1c  .h;..h.K.\.JST:.
 0020 3b1c1b68 01331360 3b1c1b68 2d2beedd  ;..h.3.`;..h-+..
 0030 0023181c bd4601b0 80bd0000 00020000  .#...F..........
 0040 30020000                             0...
Contents of section .data:
 0200 41206c6f 6e672074 696d6520 61676f2c  A long time ago,
 0210 20696e20 61206761 6c617879 20666172   in a galaxy far
 0220 2c206661 72206177 61792e2e 2e000000  , far away......
Contents of section .comment:
 0000 00474343 3a202847 4e552920 342e312e  .GCC: (GNU) 4.1.
 0010 3100                                 1.
Disassembly of section .text:
00000000 <main> b580         push    {r7, lr}
00000002 <main+0x2> b081         sub sp, #4
00000004 <main+0x4> af00         add r7, sp, #0
00000006 <main+0x6> 1c3a         adds    r2, r7, #0
00000008 <main+0x8> 2300         movs    r3, #0
0000000a <main+0xa> 6013         str r3, [r2, #0]
0000000c <main+0xc> e00c         b.n 00000028 <main+0x28>
0000000e <main+0xe> 1c3b         adds    r3, r7, #0
00000010 <main+0x10> 6819        ldr r1, [r3, #0]
00000012 <main+0x12> 1c3b        adds    r3, r7, #0
00000014 <main+0x14> 681a        ldr r2, [r3, #0]
00000016 <main+0x16> 4b09        ldr r3, [pc, #36]   (0000003c <.text+0x3c>)
00000018 <main+0x18> 5c9b        ldrb    r3, [r3, r2]
0000001a <main+0x1a> 4a09        ldr r2, [pc, #36]   (00000040 <.text+0x40>)
0000001c <main+0x1c> 5453        strb    r3, [r2, r1]
0000001e <main+0x1e> 1c3a        adds    r2, r7, #0
00000020 <main+0x20> 1c3b        adds    r3, r7, #0
00000022 <main+0x22> 681b        ldr r3, [r3, #0]
00000024 <main+0x24> 3301        adds    r3, #1
00000026 <main+0x26> 6013        str r3, [r2, #0]
00000028 <main+0x28> 1c3b        adds    r3, r7, #0
0000002a <main+0x2a> 681b        ldr r3, [r3, #0]
0000002c <main+0x2c> 2b2d        cmp r3, #45
0000002e <main+0x2e> ddee        ble.n   0000000e <main+0xe>
00000030 <main+0x30> 2300        movs    r3, #0
00000032 <main+0x32> 1c18        adds    r0, r3, #0
00000034 <main+0x34> 46bd        mov sp, r7
00000036 <main+0x36> b001        add sp, #4
00000038 <main+0x38> bd80        pop {r7, pc}
0000003a <main+0x3a> 0000        lsls    r0, r0, #0
0000003c <.text+0x3c> 0200           lsls    r0, r0, #8
0000003e <.text+0x3e> 0000           lsls    r0, r0, #0
00000040 <.text+0x40> 0230           lsls    r0, r6, #8
00000042 <.text+0x42> 0000           lsls    r0, r0, #0
```

../../project/sw/src/memcpy46.dasm

C.2 Makefile and Linker Script

```
# Example makefile for HDL Lab
# Integrated Electronic Systems Lab
# TU Darmstadt
#
# Author:   Dipl.-Ing. Boris Traskov
# Email:    boris.traskov@ies.tu-darmstadt.de
#
# Usage:
# Invoke from shell with "make <target>"
# For example, to compile count32.c run:
# make count32
# To generate utf8-encoded binary (for documentation) run:
# make text
GCC                := $(ARM_GCC_DIR)/arm-elf-gcc
SIZE               := $(ARM_GCC_DIR)/arm-elf-size
STRIP              := $(ARM_GCC_DIR)/arm-elf-strip
OBJDUMP            := $(ARM_GCC_DIR)/arm-elf-objdump

### OPTIONS SWITCHING STD-LIB FUNCTIONS ON/OFF
OPT_NOSTDLIBS      :=  -nodefaultlibs -fno-builtin -nostdlib

### OPTIONS CONTROLLING CODE GENERATION
#-mlittle-endian    Generate code for a processor running in little-endian mode.
#-mthumb or -marm   Select between generating code that executes in ARM and Thumb states.
#-O0                Reduce compilation time and make debugging produce the expected results.
#-T default.ld      specify default linker script
OPT_CODEGEN        := -mlittle-endian -mthumb -O0 -T default.ld

### OPTIONS CONTROLLING DISASSEMBLY
#-d, --disassemble          Display assembler contents of executable sections
#-s, --full-contents        Display the full contents of all sections requested
#-S, --source               Intermix source code with disassembly
#-x, --all-headers          Display the contents of all headers
#-z, --disassemble-zeroes         Do not skip blocks of zeroes when disassembling
OPT_DASM           := -dsSxz

SOURCES            := $(wildcard *.c)
TARGETS            := $(basename $(SOURCES))
BINS               := $(wildcard *.bin)
GOLDS              := $(wildcard *.gold)
LATEX_BINS         := $(patsubst %.bin, %.bintxt, $(BINS))
LATEX_GOLDS        := $(patsubst %.gold, %.goldtxt, $(GOLDS))

.PHONY: all clean $(TARGETS) text

#################################################################
#COMPILE
```

```
##################################################################
all :
    make $(TARGETS)
    make text

$(TARGETS) :
    make $@.bin $@.dasm

#used to convert binary data to utf8 for the lab manual
text : $(BINS) $(GOLDS)
    make $(LATEX_BINS) $(LATEX_GOLDS)

.SECONDARY:
%.elf : %.c
    $(GCC) $(OPT_NOSTDLIBS) $(OPT_CODEGEN)  $^ -o $@
    $(SIZE) $@
    echo $(TARGETS)

%.dasm: %.elf
    $(OBJDUMP) $(OPT_DASM) --prefix-addresses --show-raw-insn $^ > $@

%.bin: %.elf
    $(STRIP) -O binary $^ -o $@

%.bintxt : %.bin
    xxd $^ > $@

%.goldtxt : %.gold
    xxd $^ > $@
##################################################################
#REMOVE ALL INTERMEDIATE FILES
##################################################################
clean:
    rm -f *.elf *.asm *.dasm *.bin *.s *.o *txt
```

../../project/sw/src/makefile

```
SECTIONS {
    . = 0x0000;
    .text   : { *(.text) }
    . = 0x0200;
    .data   : { *(.data) }
    .bss    : { *(.bss) }
}
```

../../project/sw/src/default.ld

D Synthesis Script for Design Compiler

```
###############################################################
### SYNTHESIS SCRIPT                                        ###
###############################################################
# Integrated Electronic Systems Lab
# TU Darmstadt
# Author:   Dipl.-Ing. Boris Traskov
# Email:    boris.traskov@ies.tu-darmstadt.de
# Purpose:  map an RTL design to GL-netlist, annotate timing, analyse desgn
# Usage:    "cd" to the dc_work directory
#           invoke script with:
#           dc_shell -f ../scripts/synthesis

set PROJECT_PATH    "/home/borist/courses/hdl_lab/project"
set REP_PATH        "/home/borist/courses/hdl_lab/project/hw/rep"

#analyse, elaborate and save unmapped design
analyze -format sv {cpu.v}
elaborate cpu -architecture verilog -library WORK -update
uniquify
write -hierarchy -format ddc -output ${PROJECT_PATH}/hw/unmapped/
    cpu_unmapped.ddc

#open the schematic. Take a look at it!
#open /cad/synopsys/tools/syn_vE-2010.12-SP1/packages/gtech/src_ver/
    gtech_lib.v and take a look at it

#constrain design
create_clock -name "CLOCK" -period 1 -waveform { 0.000 0.500  }  { clk  }

#map to target technology and save
compile -exact_map
write -hierarchy -format ddc -output ${PROJECT_PATH}/hw/mapped/
    cpu_mapped.ddc

#export mapped design as verilog netlist file for gate-level simulation
write -hierarchy -format verilog -output ${PROJECT_PATH}/hw/mapped/cpu_gl.v

#export timing annotations
write_sdf ${PROJECT_PATH}/hw/mapped/timing.sdf

report_design           > ${REP_PATH}/design.txt
report_timing           > ${REP_PATH}/timing.txt
report_area             > ${REP_PATH}/area.txt
report_reference        > ${REP_PATH}/reference.txt
report_resources        > ${REP_PATH}/resources.txt
check_design            > ${REP_PATH}/check_design.txt

license_users
exit
```

../../project/hw/scripts/synthesis

E Gate-Level-Simulation Script for Modelsim

```
##################################################################
### GATE-LEVEL SIMULATION SCRIPT                               ###
##################################################################
# Integrated Electronic Systems Lab
# TU Darmstadt
# Author:   Dipl.-Ing. Boris Traskov
# Email:    boris.traskov@ies.tu-darmstadt.de
# Purpose:  simulate a verilog-netlist with annotated cell delays
# Usage:    "cd" to the ms_work directory
#           start Modelsim with:
#           VSIM $$> vsim
#           invoke GL-sim script with:
#           VSIM $$> do ../scripts/gate_level_simulation

#create a seperate library for std. cells and fill it (only needed once)
vlib umc13
vlog -work umc13 /cad/umc/UMC13/UMCE13H210D3_1.2/verilog_simulation_models/
    *

#compile gate-level netlist file and testbench
vlog -sv -work work ../mapped/cpu_gl.v ../tb/memory.sv ../tb/testbench.sv

#simulate testbench (link to standart cell library, annotate typical timing
     to Device Under Test)
vsim -L umc13 -sdftyp /testbench/cpu_i=../mapped/timing.sdf -novopt
    work.testbench

#add signals to waveform
add wave sim:/testbench/*
```

../../project/hw/scripts/gate_level_simulation

References

[1] *ARM Architecture Reference Manual.*

[2] *Design Vision HELP,*
file:///cad/synopsys/tools/syn_vE-2010.12-SP1/doc/syn/html/dvoh/enhanced/index.html.

[3] *GCC Online Documentation.*

[4] *Modelsim SE Tutorial.*

[5] *Thumb® 16-bit Instruction Set Quick Reference Card.*