

NMK206 - Computer Architecture

This course is *Computer Architecture*, offered by the Faculty of Electronics Engineering & Technology (FKTEN) for Electronic Engineering Technology programs. This page serves as an information hub, mainly for 007 lab sessions.

Download [ModelSim 20.1.1.720 \(Intel FPGA Starter Edition\) setup \(SHA-1 checksum\)](#). Linux users can try [this \(SHA-1 checksum\)](#).

Video Guide(s)

[YouTube Playlist](#)

ModelSim: Installation Note: Available on YouTube . Note: Local copy available.	 ModelSim Installation
ModelSim: Create Project Note: Available on YouTube . Note: Local copy available.	 ModelSim: Create New Project
ModelSim: Simulate Logic Circuit Note: Available on YouTube . Note: Local copy available.	 ModelSim: Simulate Logic Circuit

Announcements

[20250323] Welcome to NMK206 info page (for 007 lab sessions only)!

[20250424] Soft reminder: Lab Assessment 1 @202500420-0800!

Lab Session

I am using [Takahashi Method](#) for these slides (Actually, I broke that method by adding diagrams and long codes because I think my students need them). You will find them hard to understand if you do

not attend my sessions. So, that is the 'advantage' I gave to those who actually listen in class



- Lab Briefing
 - Slides
- Intro to CAD Tools and HDL
 - Slides
 - Online Session (Video)
- Verilog Basics
 - Slides
 - Online Session (Video)
 - [Extra]
 - Online Session (Video) (Partial... was stopped due to low number of students)
- Combinational Logic
 - Slides (P1)
 - Slides (P2)
 - Slides (P3)
 - Slides (P4)
 - Slides (P5)
- Sequential Logic
 - Slides (P1)
 - Slides (P2)
- State Machine
 - Slides
- Simple Digital System
 - Slides

Verilog Coding Rule

This is the coding rule that I impose on my students. You will be penalized during assessments if it is not adhered to.

1. One file for one module
 - **RULE:** 1 file 1 module
2. **File name must be the same as module name**
 - **RULE:** file name === module name (.v)
3. All circuit (module) must have a testbench (tb)
 - tb is a also module (so, separate file)
 - **RULE:** All module MUST have a testbench
 - **RULE:** Tb name === module name + _tb
4. Use Verilog95 module declaration
 - Port list contain names only (separate input/output declaration)
 - Port connection(s) MUST BE specified using ordered list
 - **RULE:** Port connection(s) by ordered list ONLY!
5. Modules for combinational logic should only use wire/assign statements

- reg/always reserved for sequential logic and testbench modules
- **RULE:** comb. logic use assign/wire only!
- 6. Only basic logic gates are allowed
 - Can only use AND/OR/INV in your logic implementation
 - XOR logic is allowed for **lab project only**
 - **RULE:** allowed operators AND, OR, INV
- 7. Assign statements can only have ONE binary operator
 - Multiple bitwise inverts (~) are ok (they are unary operators)
 - **RULE:** 2-input logic gates ONLY
- 8. ALL nets (wire/reg) MUST BE declared.
 - some compiler may allow using without declaration
 - for my assessments, they MUST BE declared
 - **RULE:** All signals @wire must be declared!

Lab Code (202425s2) Archive

[Complete template for a simple 4-bit processor core \(as discussed during class\) available](#)

here

Lab Code (202425s2) Extra

Here is an example code to utilize the carry/borrow signals of our add/subtract modules at ALU level. You may use this to create add-with-carry and subtract-with-borrow functions (hint: you need a selector signal to 'enable' the carry/borrow in signals {iC} and {iB}).

[alu_flag_4b.v](#)

```
module alu_4b (iS,iP,iQ,iF,oF,oY);
input[1:0] iS;
input[3:0] iP,iQ,iF;
output[3:0] oF,oY;
wire[3:0] oY;
wire iC, iB, oC,oB;

wire[3:0] tA0, tA1;
wire[3:0] tL0, tL1;

add_4b a_add (iC,iP,iQ,oC,tA0);
sub_4b a_sub (iB,iP,iQ,oB,tA1);
and_4b l_and (iP,iQ,tL0);
or_4b l_or (iP,iQ,tL1);

// separate flag bits for borrow and carry
assign oF = { 2'b00 , oB, oC };
// should get from same bit position
assign iB = iF[1];
assign iC = iF[0];
```

```
dmux41 sel0 (iS,tA0,tA1,tL0,tL1,oY);

endmodule
```

alu_flag_4b_tb.v

```
module alu_4b_tb ();
reg[1:0] dS;
reg[3:0] dP,dQ,dF;
wire[3:0] mF,mY;
reg[3:0] cF,cY;

integer loop, ecnt;

initial begin
    ecnt = 0;
    // TASK: try to put dS assignment in a loop! => shorter code!
    dS = 2'b00; // test add
    for (loop=0;loop<512;loop++) begin
        {dF[0],dP,dQ} = loop;
        #5;
        { cF[0],cY } = dF[0] + dP + dQ;
        if (cY!=mY||cF[0]!=mF[0]) begin
            ecnt = ecnt + 1;
            $display("*** Add error
(%x+%x+%x=%x@%x| %x:%x)!",dP,dQ,dF[0],mY,cY,
                mF[0],cF[0]);
        end
    end
    dS = 2'b01; // test sub
    for (loop=0;loop<512;loop++) begin
        {dF[1],dP,dQ} = loop;
        #5;
        { cF[1],cY } = dP - dQ - dF[1];
        if (cY!=mY||cF[1]!=mF[1]) begin
            ecnt = ecnt + 1;
            $display("*** Sub error (%x-%x-%x=%x@%x)!",dP,dQ,dF[1],mY,cY,
                mF[1],cF[1]);
        end
    end
    dS = 2'b10; // test and
    for (loop=0;loop<256;loop++) begin
        {dP,dQ} = loop;
        #5;
        cY = dP & dQ;
        if (cY!=mY) begin
            ecnt = ecnt + 1;
            $display("*** And error (%x&%x=%x@%x)!",dP,dQ,mY,cY);
        end
    end
end
```

```

end
dS = 2'b11; // test or
for (loop=0;loop<256;loop++) begin
    {dP,dQ} = loop;
    #5;
    cY = dP | dQ;
    if (cY!=mY) begin
        ecnt = ecnt + 1;
        $display("** 0r error (%x|%x=%x@%x)!",dP,dQ,mY,cY);
    end
end
if (ecnt==0) begin
    $display("-- Module alu_4b verified!");
end
else begin
    $display("** Module alu_4b with %g error(s)!",ecnt);
end
end
alu_4b dut (dS,dP,dQ,dF,mF,mY);

endmodule

```

Lab Project (202425s2) Requirements

This is also shared in Google Doc format (link available in Google Classroom).

[nmk206-202425s2_labproject.txt](#)

```

-----
LAB PROJECT
-----

```

You are required to implement a soft-processor core (HDL-based) with the following minimum requirements:

- 8-bit microprocessor (instruction size, register size, etc.)
- 8 general purpose registers
- 8 ALU functions
- basic instruction set
 - = move data between registers
 - = perform basic ALU operations
 - = load register with immediate value

These requirements are necessary for submission and minimum grade B. More functionality means better grades:

- 16-bit/32-bit microprocessor
- carry circuit for adder (>8b)

- integer multiply/divide circuit (structural code - NOT behavior)
- bit-level manipulation (towards microcontroller)
- instruction fetch unit and memory interface

Note#1: Auto-0 for downloaded codes!

Note#2: This is a group assignment, but marks will be evaluated individually.

Assessment deliverables:

1) Verilog source files in a single ZIP file

- make sure only Verilog files are included in the file
- make sure all modules have each a proper testbench
- must be self-checking testbenches

2) Technical Document (Specifications)

- not more than 10 pages, no cover page (only list of members' names)
- content:
 - = short summary of features
 - = block diagram of design
 - = list of instructions (description and op-code)

3) An online demonstration / Q&A

- not more than 20 minutes
- content:
 - = short summary of what has been implemented
 - = if completed, demonstrate running the top-level testbench
 - = if NOT completed, demonstrate running component-level testbenches

PROJECT DUE: Lab Session@W14/14 (Item 1&2). Item 3 TBD.

Extra Info:

- each student may email in a contribution percentage information:
 - = list all group members (specify project contribution percentage of each)
 - = percentage values should total up to 100% (may use fraction e.g. 1/3)

Note: this is optional, but if a member decides to do this, please advise

the others in group to do the same

From:

<https://azman.my1matrix.cc/> - Azman @MY1

Permanent link:

<https://azman.my1matrix.cc/doku.php?id=archive:nmk206>

Last update: **2026/02/08 03:46**



