Aalto University
School of Electrical
Engineering

ELEC-E3540 Digital Microelectronics II (5cr)
IV-V period
Introduction

Marko Kosunen

Department of Micro and Nanosciences
Aalto University, School of Electrical Engineering
marko.kosunen@aalto.fi

March 2, 2021

Outline

On Digital design

Design implementation methods

- Logic gates are designed using transistors on device (transistor) level.
- Simple logic functions can be designed with truth tables or Karnaugh maps on gate level, although it is beneficial to synthesize the blocs with automated design tools.
- More complex functions/algorithm implementations or entire systems are modeled with Hardware Description Languages which are used together with a set of automated design tools.
- The actual implementation is performed by the tools, so ability to control the tools efficiently is mandatory.
- Current effort is to move to the higher abstraction level while designing (behavioral synthesis from VHDL or System C).

Digital hardware design

$$F = \overline{A \cdot B}$$

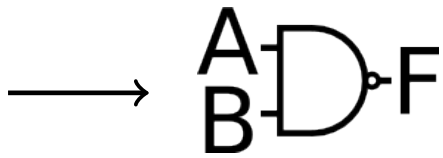
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

```
F<= NOT ( A AND B);
```

Digital hardware design

$$F = \overline{A \cdot B}$$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

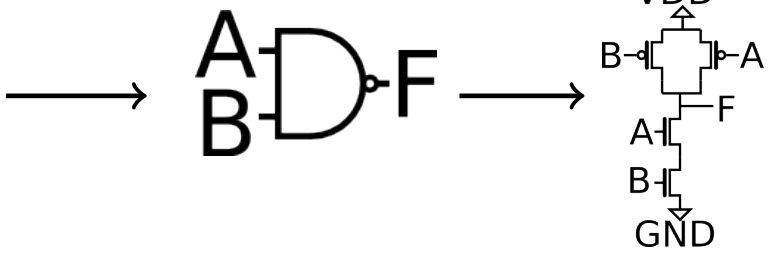



```
F<= NOT ( A AND B);
```

Digital hardware design

$$F=\overline{A \cdot B}$$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0



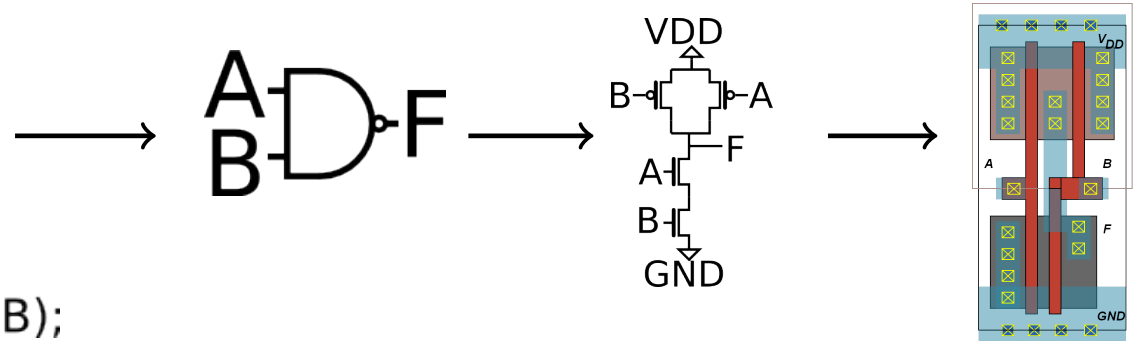

```
F<= NOT ( A AND B);
```

Digital hardware design

$$F = \overline{A \cdot B}$$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

F<= NOT (A AND B);

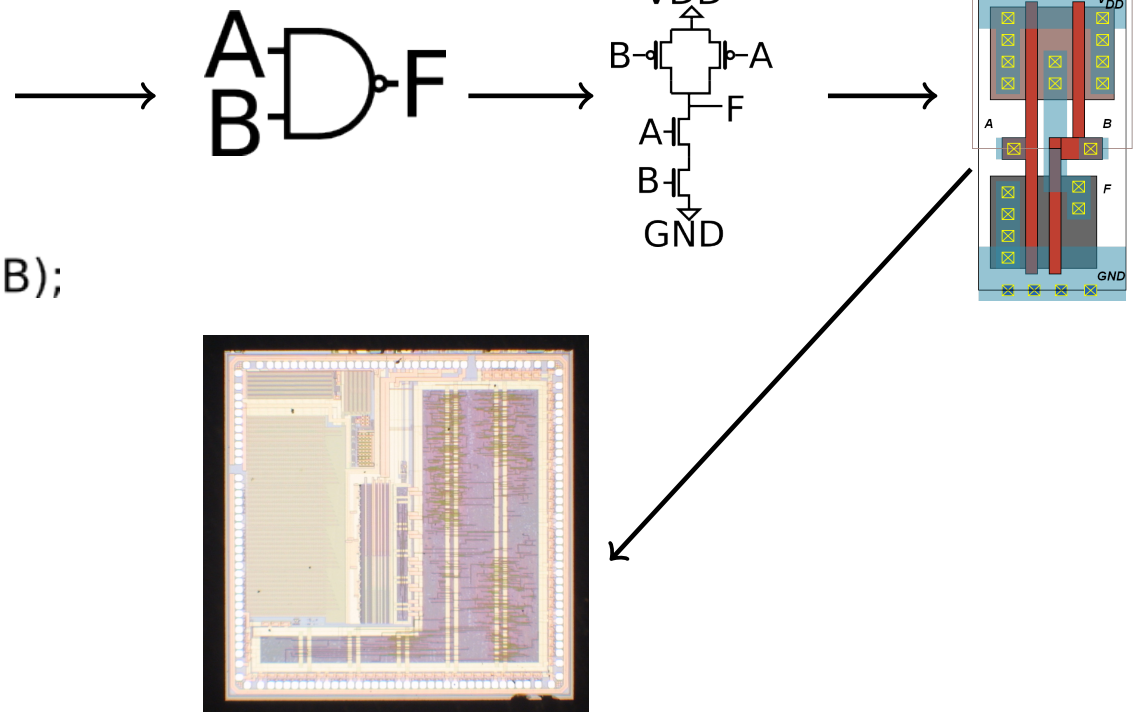


Digital hardware design

$F = \overline{A \cdot B}$

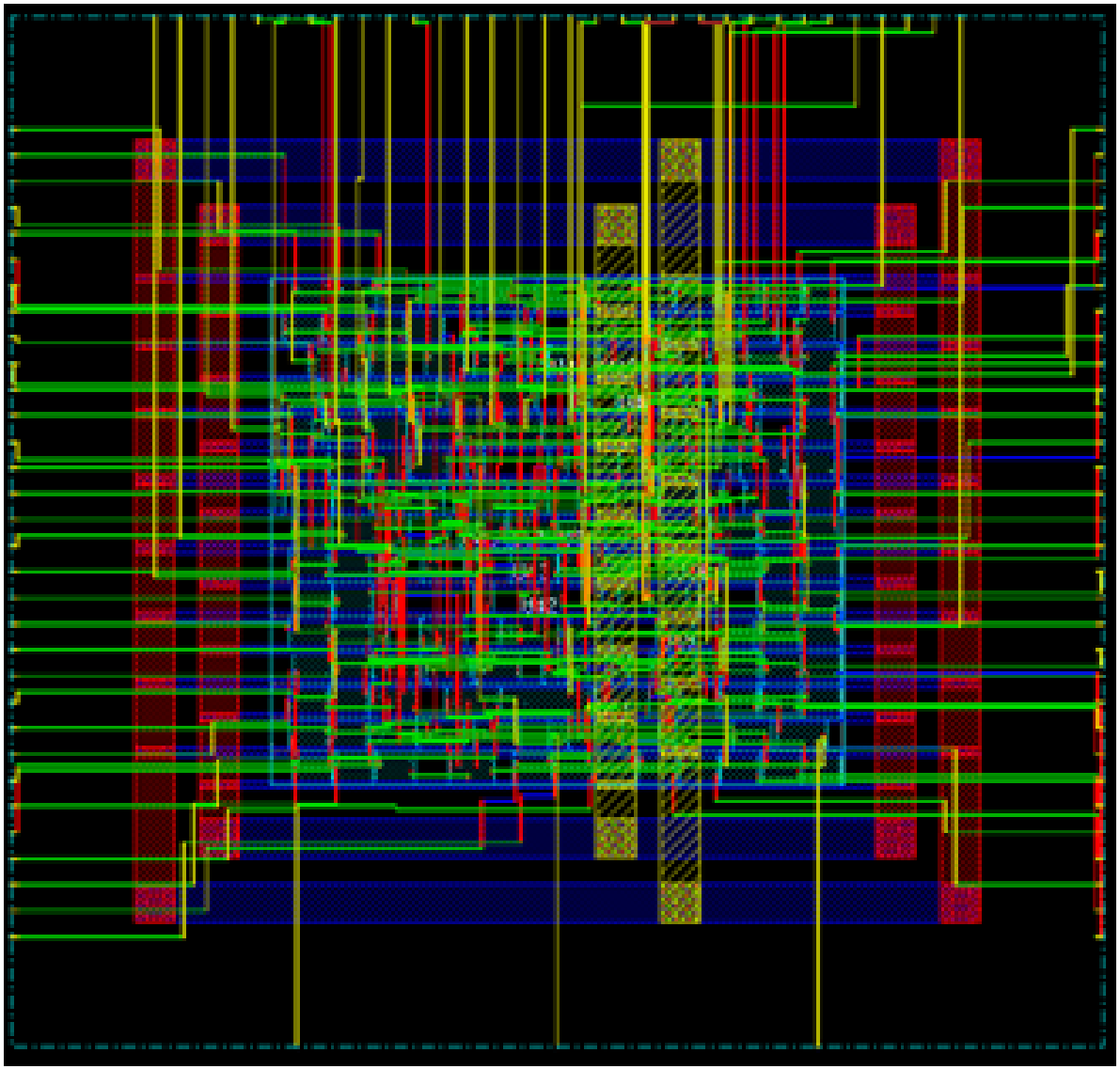
A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

F<= NOT (A AND B);

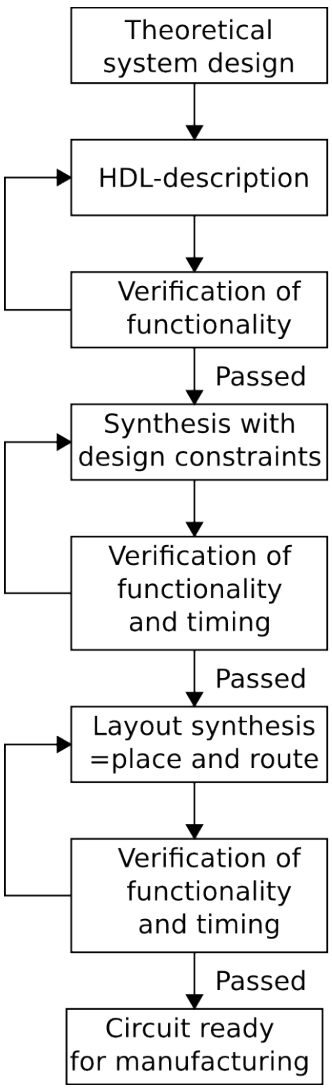


Designing digital circuits is mapping logical functions to transistor level equivalents, *implemented* on a chosen platform, ASIC or FPGA.

Layout-the design database for manufacturing



Synthesis Flow



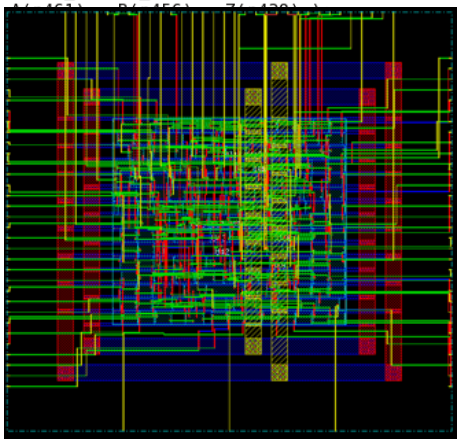
→ This has been done for you already :)

$F=\overline{A \cdot B}$

A	B	F
0	0	1
0	1	1
1	0	1
1	1	0

```
F<= NOT ( A AND B);
```

```
HS65_LH_DFPRQX9 gray_counter_m_generate_MSB_count_reg ( .D(
      gray_counter_m_generate_MSB_N0), .CP(osc_inm), .RN(n445), .Q(
      gray_counter_m_gray_count_16_) );
HS65_LH_MUX21X4 U247 ( .D0(generate_piso_onehotv_1_), .D1(
      generate_piso_onehotv_32_), .S0(n452), .Z(n216) );
HS65_LH_IVX7 U262 ( .A(gray_counter_p_gray_count_0_), .Z(
      gray_counter_p_generate_parity_N0) );
HS65_LH_IVX7 U263 ( .A(gray_counter_m_gray_count_0_), .Z(
      gray_counter_m_generate_parity_N0) );
HS65_LH_NAND2X4 U265 (
```



Course arrangements

Your teachers

- **D.Sc. Marko Kosunen** marko.kosunen@aalto.fi “lecturer”, TUAS-2190
- **M.Sc. student Andrei Spelman**, andrei.spelman@aalto.fi, assistant.
- D.Sc. Vishnu Unnikrishnan, vishnu.unnikirishnan@aalto.fi, assistant, TUAS-2189.
- Most of the support is provided during the exercise sessions on Mondays 10-12 or Thursdays 10-12 in Slack and Zoom.
- On this we do not have classroom teaching. All teaching is executed remotely.

Course objective

- Objective is to learn to implement digital circuits *on higher abstraction level than the transistor*
- Modeling of complex functions/algorithms or entire systems with hardware description language (HDL)
- Translation into gate-level netlist and circuit layout with automated synthesis and place-and-route software tools
- Consider the following:
  - Examples of digital circuits, what are they?
  - How the digital circuits are designed and implemented?
  - Examples of design methods?
  - Examples of implementation methods?

Course structure

- **This is a self-learning course**: there will be only one lecture besides this one.
- Of course, help will be provided upon request
- **Material**:
  - To finish the exercises, you need a book: Peter J Ashenden, “The designer’s guide to VHDL”, 3rd edition.
  - Slides, tutorials, instructions, etc. available in MyCourses and Aalto version

Course structure

- **Six mandatory pass/fail graded exercises**
  - Topics of the exercises are given in Aalto Version.
  - For every exercise, a “pre-exercise task” is given in order to prepare yourself for the actual exercise time.
- **Design assignment**: implementation of PIC16F84A microcontroller
  - Learn the complete design flow of a complex digital system (VHDL + synthesis + place-and-route)
  - Final course grade = design assignment grade

Course rules and schedule

- Purpose of exercise sessions:
  - Main time to ask for help
  - During the exercise time, you should ask for help for the problematic parts, and finish the exercise.
  - If questions outside exercise sessions should be posted to slack.
  - Excellent time to “return” completed exercises
  - Returning outside exercise session is possible, if your testbench can be ran with a single command, and the simulation is flawless.
  - In this case, the exercise is returned as a Gitlab issue, and assigned to assistant for review.
  - Do not send emails, unless the question is related to personal matter or course administration.

Course rules and schedule

- Exercises can be time-consuming, so exercise session times are not sufficient you must work also independently between the sessions
- Exercises must be returned in order
  - Not possible to e.g. return exercise 2 before 1

How to pass

- Complete all six exercises and get them accepted by the teacher or assistant
- Complete the design assignment and submit it via MyCourses (Code should be submitted to Aalto Version)
- Firm deadline for the project and everything is: 31st May 2018

Content of mandatory exercises

Six mandatory exercises

- Exercise will be carried out and completed during a Slack Hack session.
- Assistant/lecturer will be there to help you and accept your exercise to be completed.
- Time is limited, therefore, completing the pre-exercise task beforehand is recommended.
- Exercises will be performed through X2Go NX-client and ssh from computer class to computing machine vspace of Department of MNT. Computer accounts required.
- Connection accessible only from Aalto network (e.g., computer classrooms and through VPN)
- Tools to be used: git, gvim or emacs, Mentor Graphics Questasim.

On text editors

- The text editor is the most important tool you will ever use.
- I strongly suggest that you choose vim (or emacs) as your text editor.
  - Decent setup for vim will be provided with *skeleton* git project at *https://version.aalto.fi/gitlab/elec-e3540-exec/skeleton*
  - Go through “gvimtutor” to learn the basics.

Exercise topics

- **Exercise 1:** Test benches. Book chapters 1.1-1.4, 2.1-2.2, 2.5, 5.1-5.2 .
  Things to learn: Types: bit, bit_vector, boolean, integer. Signals, process, variables, sensitivity lists of the processes, process as a part of a test bench, wait-statement, self-terminating simulation.
  Workload: 2+4=6h
- **Exercise 2:** For loops and file IO, Book chapters 3.4, 5.3,13.1, 16.1
  Things to learn: Components, File-IO, For loops, Using the previously written test bench.
  Workload: 2+4=6h

Exercise topics

- **Exercise 3:** Book chapters 9.1-9.2, 4.1-4.4, 14.1-14.2
  Things to learn: Libraries, Vectors and arrays, Records, for-if-generate, Indexing
  Workload: 2+4=6h
- **Exercise 4:** Operation decoder for PIC. Book chapters 3.1-3.5, 2.2.5, 5.2, 21.5
  Things to learn: State machines, Edge sensitive processes, Synchronous logic, if and case, assert.
  Workload: 2+8=10h

Exercise topics

- **Exercise 5:** ALU Design. Book chapters 6.1-6.6
  Things to learn: Procedures and functions, structure of ALU of the PIC16F84A microcontroller.
  Workload: 2+8=10h
- **Exercise 6:** Memory design. Book chapters 21 particularly 21.6.
  Things to earn: Memory implementations, Design for synthesis
  Workload: (about) 2+8=10h
- Requirement for passing the course is participation and accepted results of the exercises. Pass/Fail grading.

Design assingment: Microcontroller implementaion

Design assingment: Microcontroller implementaion

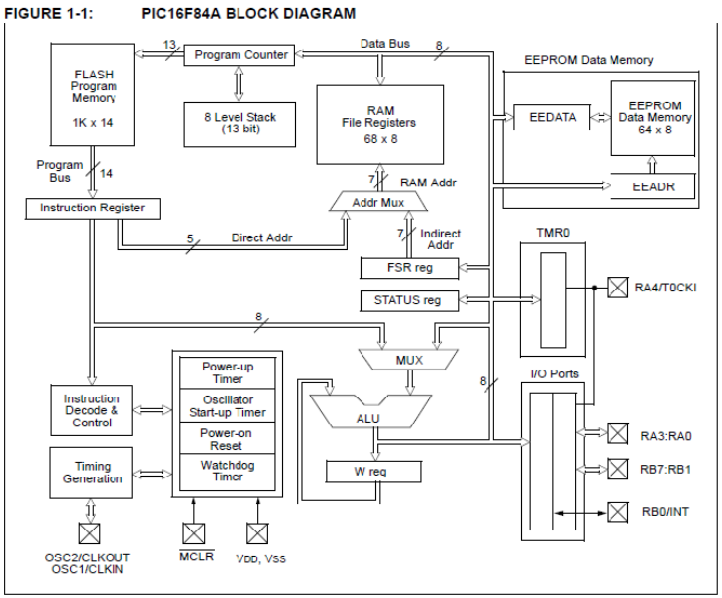
- Perform the whole digital IC implementation flow of part of the PIC16F84A microcontroller VHDL modeling + synthesis
- Starting year 2021, Place and route will be separated to course ow it's own.
- PIC chosen because of its simple structure, and because assembler compiler is available
- Nevertheless, learning its functionality is not very straightforward, so start studying immediately! datasheet available in MyCourses

Design Assignment: Microcontroller implementation with VHDL

- Course will be graded based on study diary and documentation of the design
- the study diary should document and describe the phases of the design flow, difficulties encountered and how they were solved
- Things to be graded:
  - Quality of the code, clear structure, commented, easy to read.
  - Gained understanding of the subject. This should be visible in your study diary.
  - 100% functionality is not required to pass, but you should show that you have tried your best and learned something.

Design Assignment: Microcontroller implementation with VHDL

- On this course you will learn VHDL and the whole digital IC implementation flow while designing a PIC16F84A Microcontroller.
- You learn the required skills during the first six exercises, contents of which also supports the PIC design task.
- Learning the functionality of PIC is hard, so start studying the PIC-datasheet immediately.



;

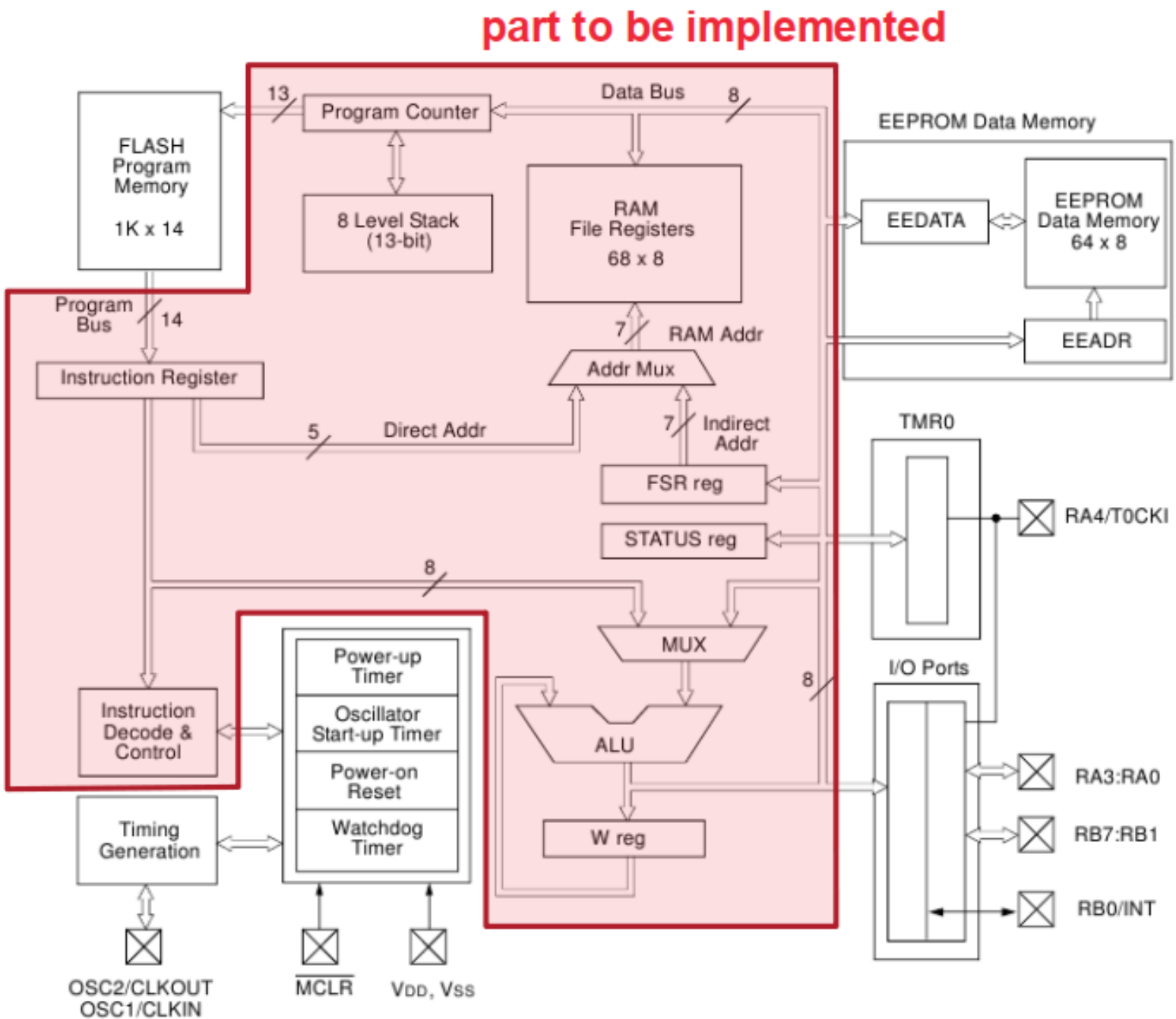
Microcontroller implementation

- PIC16F84A has been chosen because
  - Simple structure
  - Small instruction set
  - Simple ALU
  - Assembler compiler available

Microcontroller implementation

- Design process of the microcontroller should be documented in a study diary describing the design process, methods, difficulties and sources of information.
- The designed microcontroller will be also synthesized to logic.
- Things to be graded
  - Quality of the code, clear structure, commented, easy to read.
  - Understanding of the subject gained. This should be visible in your study diary.
  - 100% functionality is not a required, but you should show that you have tried your best and learned something.
  - The grade of the PIC design assignment is the grade of the course.

Microcontroller implementation



Microcontroller implementation

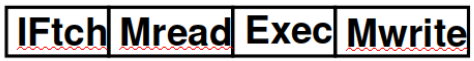
instructions NOT to be implemented

Mnemonic, Operands		Description	Cycles	14-Bit Opcode MSb			LSb	Status Affected	Notes
BYTE-ORIENTED FILE REGISTER OPERATIONS									
ADDWF	f, d	Add W and f	1	00	0111	dfff	ffff	C,DC,Z	1,2
ANDWF	f, d	AND W with f	1	00	0101	dfff	ffff	Z	1,2
CLRF	f	Clear f	1	00	0001	lfff	ffff	Z	2
CLRW	-	Clear W	1	00	0001	0xxx	xxxx	Z	
COMF	f, d	Complement f	1	00	1001	dfff	ffff	Z	1,2
DECF	f, d	Decrement f	1	00	0011	dfff	ffff	Z	1,2
~~DECFSZ~~	~~f, d~~	~~Decrement f, Skip if 0~~	1 (2)	00	1011	dfff	ffff		1,2,3
INCF	f, d	Increment f	1	00	1010	dfff	ffff	Z	1,2
~~INCFSZ~~	~~f, d~~	~~Increment f, Skip if 0~~	1 (2)	00	1111	dfff	ffff		1,2,3
IORWF	f, d	Inclusive OR W with f	1	00	0100	dfff	ffff	Z	1,2
MOVF	f, d	Move f	1	00	1000	dfff	ffff	Z	1,2
MOVWF	f	Move W to f	1	00	0000	lfff	ffff		
NOP	-	No Operation	1	00	0000	0xx0	0000		
RLF	f, d	Rotate Left f through Carry	1	00	1101	dfff	ffff	C	1,2
RRF	f, d	Rotate Right f through Carry	1	00	1100	dfff	ffff	C	1,2
SUBWF	f, d	Subtract W from f	1	00	0010	dfff	ffff	C,DC,Z	1,2
SWAPF	f, d	Swap nibbles in f	1	00	1110	dfff	ffff		1,2
XORWF	f, d	Exclusive OR W with f	1	00	0110	dfff	ffff	Z	1,2
BIT-ORIENTED FILE REGISTER OPERATIONS									
BCF	f, b	Bit Clear f	1	01	00bb	bfff	ffff		1,2
BSF	f, b	Bit Set f	1	01	01bb	bfff	ffff		1,2
~~BTFSC~~	~~f, b~~	~~Bit Test f, Skip if Clear~~	1 (2)	01	10bb	bfff	ffff		3
~~BTFSS~~	~~f, b~~	~~Bit Test f, Skip if Set~~	1 (2)	01	11bb	bfff	ffff		3
LITERAL AND CONTROL OPERATIONS									
ADDLW	k	Add literal and W	1	11	111x	kkkk	kkkk	C,DC,Z	
ANDLW	k	AND literal with W	1	11	1001	kkkk	kkkk	Z	
~~CALL~~	~~k~~	~~Call subroutine~~	2	10	0kkk	kkkk	kkkk		
~~CLRWDT~~	~~-~~	~~Clear Watchdog Timer~~	1	00	0000	0110	0100	$\overline{TO}$,$\overline{PD}$	
~~GOTO~~	~~k~~	~~Go to address~~	2	10	1kkk	kkkk	kkkk		
IORLW	k	Inclusive OR literal with W	1	11	1000	kkkk	kkkk	Z	
MOVLW	k	Move literal to W	1	11	00xx	kkkk	kkkk		
~~RETFIE~~	~~-~~	~~Return from interrupt~~	2	00	0000	0000	1001		
~~RETLW~~	~~k~~	~~Return with literal in W~~	2	11	01xx	kkkk	kkkk		
~~RETURN~~	~~-~~	~~Return from Subroutine~~	2	00	0000	0000	1000		
~~SLEEP~~	~~-~~	~~Go into standby mode~~	1	00	0000	0110	0011	$\overline{TO}$,$\overline{PD}$	
SUBLW	k	Subtract W from literal	1	11	110x	kkkk	kkkk	C,DC,Z	
XORLW	k	Exclusive OR literal with W	1	11	1010	kkkk	kkkk	Z	

;

Advice: Stages of command execution

- **IFETCH**: Fetch instruction from program memory and decode it.
- **Mread**: Read operand from memory, if required.
- **Execute**: Perform operation.
- **Mwrite**: Increment PC, write data to memory or register.



;

- Every instruction can be divided in “stages”. Maximum number is four, since PIC datasheet describes execution in max four clock cycles.
- Only Mwrite is strictly synchronous operation, but in order to make things easier, advice is to implement the steps with a synchronous state machine.
- Every command does not require every step.

Conlusion and next steps

Course Feedback

In the beginning of the course I had never even seen VHDL code before and we were warned that this course would take alot of time. This did not turn out to be an overstatement. Only the major developments and complications were reported in this diary, since if I would have given full disclosure this diary would probably be double or triple the current length. Although, this course was one of the most exhausting courses I have been to, I did learn alot of things.

Course Feedback

This project has been quite a challenge since, every thing was done for the first time. At the start of this course, I completely had zero idea on VHDL, and digital flow. At the end, I am very confident to try out challenging tasks and set my career at digital design field. Overall, although quite hectic workload, I am completely satisfied with what I have achieved.

Course Feedback

Lets also summarize some other aspects learned during the process :

- I have advanced in Vim editor environment to some extent . It made re-writing and debugging my code very easy. I insist on every designer to know to use any editor very well.
- VHDL codes are not like the programming. The way to think is to think hardware. Same structure can be implemented using different commands, but it will effect how RTL is synthesized. Its upto designer to choose command to best fit performance.
- While writing HDL codes, insted of giving fancy complicated code, its more healthy habit to follow basic synthesizable template.

Course Feedback

- Digital design flow is very automatic process. The software will do the routing and generate reports. So its necessary to know how to use the software so that correct results are obtained.
- Digital design is very lengthy type of repetitive process, to prevent loss of enormous time and frustation, TCL scripting knowledge will prove to be very handy
- Another point would I emphasize while on starting to write HDL is to proper test bench. Once made so that it will fit the purpose of whole project, it shall save lot of time.
- Proper use of case statement and If else statement is very tricky in VHDL. Deeper understanding in use of these statements can lead to more optimized design

How to start?

- Establish X2Go connection to vspace.ecdl.hut.fi
- Add your vspace ssh key to Aalto version *https://version.aalto.fi/gitlab/profile/keys* Instructions are also provided there.
- Go to *https://version.aalto.fi/gitlab/elec-e3540-exec/skeleton* , read the Readme.
- Clone it to your home directory with *git clone gitversion.aalto.fi:elec-e3540-exec/skeleton.git* and do the setups as instructed in the readme.
- Go to *https://version.aalto.fi/gitlab/elec-e3540-exec/exercise_template*, and read the Readme.

Appendix: Very short and comprehensive VHDL guideline

Very short and comprehensive VHDL guideline

- Dedicated directory for the VHDL code.

Very short and comprehensive VHDL guideline

- Dedicated directory for the VHDL code.
- File name always the same as entity name.

Very short and comprehensive VHDL guideline

- Dedicated directory for the VHDL code.
- File name always the same as entity name.
- Architecture always in the same file as the entity.

Very short and comprehensive VHDL guideline

- Dedicated directory for the VHDL code.
- File name always the same as entity name.
- Architecture always in the same file as the entity.
- Simple architecture names ’behav’-for behavioral and ’rtl’-for synthesizable code are sufficient. No need to use imagination here.

Very short and comprehensive VHDL guideline

- Dedicated directory for the VHDL code.
- File name always the same as entity name.
- Architecture always in the same file as the entity.
- Simple architecture names ’behav’-for behavioral and ’rtl’-for synthesizable code are sufficient. No need to use imagination here.
- Proper intendation, commenting, and structuring of the code, as in any coding.

Very short and comprehensive VHDL guideline

- Dedicated directory for the VHDL code.
- File name always the same as entity name.
- Architecture always in the same file as the entity.
- Simple architecture names ’behav’-for behavioral and ’rtl’-for synthesizable code are sufficient. No need to use imagination here.
- Proper intendation, commenting, and structuring of the code, as in any coding.
- No components and processes in the same file. Looks messy. Testbench is an exception (no strict quideline).
- No positional port assignment.

File info and licensing

```
--------------------------------------------------------------------------------
-- Copyright (c) 2016  Aalto University
--
-- Permission is hereby granted, free of charge, to any person obtaining a copy
-- of this software and associated documentation files (the "Software"), to deal
-- in the Software without restriction, including without limitation the rights
-- to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
-- copies of the Software, and to permit persons to whom the Software is
-- furnished to do so, subject to the following conditions:
--
-- Licensee is not allowed to distribute the Software by making it publicly
-- available.
--
-- The above copyright notice and this permission notice shall be included in all
-- copies or substantial portions of the Software.
--
-- THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
-- IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
-- FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
-- AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
-- LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
-- OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
-- SOFTWARE.
--------------------------------------------------------------------------------
--
-- Timing delay measurement apparatus
-- Initially written by Marko Kosunen
--
-- Last modified by 20160219 17:50 marko.kosunen@aalto.fi
--------------------------------------------------------------------------------
--
-- Generics:
-- CycleCounterBits :   Defines the maximum of measurement duration
--                      in lo cycles.
-- nGates           :   How many register stages are used in clk domain
--                      Boundaries to alleviate metastability problems.
--
-- Ports:
-- rst              :   Master reset.
-- lo_div_in            :   lo_signal, also clk signal. it is assumed, that
--                      the delay measurement is to be perfromed once
--                      during the cycle of this signal.
--                      Usually a master lo divided by some factor.
-- osc_in           :   Oscillator input. This signal is not in synchronony with
--                      lo_div_in. It is not allowed to be in synchrony.
-- osc_in_div       :   If the osc_in is too fast, it is possible to
--                      divide input osc freq by 1,2, 4 or 8.
-- start            :   Rising edge of this between two rising lo_div_in's triggers
--                      the measurement.
-- ncycles          :   How many cycles will be measured.
-- A_ref            :   Reference point of the measurement. Rising edge enables
--                      the measurement counter.
-- A_del            :   Delayed A_ref. Rising edge disables the measurement counter.
-- count_ref        :   How many osc periods the measurement lasted.
--                      Together with ncycles, provides the timing reference
-- count_meas       :   Relatively, how long is the pulse A_ref->A_del
--                      compared to lo_div_in period.
-- meas_rdy         :   Measurement ready indicator. Data at the output registers
--                      is valid when this signal is high.
--------------------------------------------------------------------------------
```

Library definitions

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE std.textio.ALL;
```

Entity declaration

```
ENTITY timing_difference_meter IS
    GENERIC(CycleCounterBits : INTEGER := 30;
            nGates           : INTEGER := 4 );
    PORT(rst          : IN  STD_LOGIC;
         lo_div_in    : IN  STD_LOGIC;
         osc_in       : IN  STD_LOGIC;
         osc_in_div   : IN  STD_LOGIC_VECTOR(1 DOWNTO 0);
         start        : IN  STD_LOGIC;
         ncycles      : IN  STD_LOGIC_VECTOR(CycleCounterBits-1 DOWNTO 0);
         A_ref        : IN  STD_LOGIC;
         A_del        : IN  STD_LOGIC;
         count_ref    : OUT STD_LOGIC_VECTOR(CycleCounterBits+1 DOWNTO 0);
         count_meas   : OUT STD_LOGIC_VECTOR(CycleCounterBits+1 DOWNTO 0);
         meas_rdy     : OUT STD_LOGIC
        );
END timing_difference_meter;
```

Architecture, type and signal declaration

```
ARCHITECTURE rtl OF timing_difference_meter IS
    TYPE timing_meas_state IS (idle, rst_counts, running,writing);
    SIGNAL timing_meas_state_current: timing_meas_state;
    SIGNAL prev_start                : STD_LOGIC;
    SIGNAL cycle_counter             : STD_LOGIC_VECTOR(CycleCounterBits-1 DOWNTO 0);
    SIGNAL s_en_ref                  : STD_LOGIC;
    SIGNAL s_en_meas_aref            : STD_LOGIC;
    SIGNAL s_en_meas_adel            : STD_LOGIC;
    SIGNAL s_en_meas                 : STD_LOGIC;
    SIGNAL s_rst_counters            : STD_LOGIC;
    SIGNAL s_meas_rdy                : STD_LOGIC;
    SIGNAL s_en_ref_gated            : STD_LOGIC_VECTOR(0 TO nGates);
    SIGNAL s_en_meas_gated           : STD_LOGIC_VECTOR(0 TO nGates);
    SIGNAL s_meas_rdy_gated          : STD_LOGIC_VECTOR(0 TO nGates);
    SIGNAL s_write_regs_gated        : STD_LOGIC_VECTOR(0 TO nGates );
    SIGNAL s_meas_rdy_gated_lo       : STD_LOGIC_VECTOR(0 TO nGates);
    SIGNAL s_write_regs_gated_lo     : STD_LOGIC_VECTOR(0 TO nGates );

    SIGNAL s_osc_in                  : STD_LOGIC;

    SIGNAL s_osc_div_p2              : STD_LOGIC;
    SIGNAL s_osc_div_p4              : STD_LOGIC;
    SIGNAL s_osc_div_p8              : STD_LOGIC;
    --2 more bits for measurement counters allow faster oscilllator than lo
    SIGNAL s_count_ref               : STD_LOGIC_VECTOR(CycleCounterBits+1 DOWNTO 0);
    SIGNAL s_count_meas              : STD_LOGIC_VECTOR(CycleCounterBits+1 DOWNTO 0);
    SIGNAL s_write_regs              : STD_LOGIC;
    SIGNAL s_ack_write_regs          : STD_LOGIC;
    SIGNAL cycleone                  : STD_LOGIC_VECTOR(CycleCounterBits-1 DOWNTO 0);
    SIGNAL counterone                : STD_LOGIC_VECTOR(CycleCounterBits+1 DOWNTO 0);
BEGIN
--END rtl
```

Signal assignment

```
--Constants
cycleone(CycleCounterBits-1 DOWNTO 1)<=(OTHERS=>'0');
cycleone(0)<='1';

counterone(CycleCounterBits+1 DOWNTO 1)<=(OTHERS=>'0');
counterone(0)<='1';
```

Processes and variables

```
--The osc_in_divider
oscDiv:PROCESS(osc_in,osc_in_div,rst)
    VARIABLE v_osc_div_p2          : STD_LOGIC;
    VARIABLE v_osc_div_p4          : STD_LOGIC;
    VARIABLE v_osc_div_p8          : STD_LOGIC;
    VARIABLE v_osc_in              : STD_LOGIC;
BEGIN
    IF (rst='1') THEN
        v_osc_div_p2:='0';
        v_osc_div_p4:='0';
        v_osc_div_p8:='0';
        v_osc_in:='0';
    ELSIF (rst='0') THEN
        IF RISING_EDGE(osc_in) THEN
            v_osc_div_p2:=NOT s_osc_div_p2;
            IF (s_osc_div_p2 = '0') THEN
                v_osc_div_p4:=NOT s_osc_div_p4;
            END IF;
            IF ((s_osc_div_p2 OR s_osc_div_p4) = '0') THEN
            v_osc_div_p8:=NOT s_osc_div_p8;
             END IF;
        END IF;
    END IF;
    CASE osc_in_div IS
        WHEN "00" =>
            v_osc_in:=osc_in;
        WHEN "01" =>
            v_osc_in:=v_osc_div_p2;
        WHEN "10" =>
            v_osc_in:=v_osc_div_p4;
        WHEN "11" =>
            v_osc_in:=v_osc_div_p8;
        WHEN OTHERS =>
            v_osc_in:=osc_in;
    END CASE;
    s_osc_div_p2<=v_osc_div_p2;
    s_osc_div_p4<=v_osc_div_p4;
    s_osc_div_p8<=v_osc_div_p8;
    s_osc_in<=v_osc_in;
END PROCESS;
```

Component usage

```
--Dut here
measurementUnit:timing_difference_meter
    GENERIC MAP(Bits      => Bits,
                nGates    => nGates)
    PORT MAP(
        rst         => rst,
        lo_in       => lo,
        osc_in      => osc,
        start       => start,
        ncycles     => ncycles,
        A_ref       => A_ref,
        A_del       => A_del,
        count_ref   => count_ref,
        count_meas  => count_meas,
        meas_rdy    => meas_rdy
       );
```