ELEC-E3540 Digital Microelectronics II

Assignment instructions

Enrico Roverato

February 2018

Aalto University
School of Electrical
Engineering

Contents

1 General information

The purpose of this assignment is to teach the student:

1. The basics of the microcontrollers and their functionality by implementing part of a PIC16F84A microcontroller with VHDL.
2. How to design and describe digital microelectronic circuits by using synthesizable VHDL.
3. The basic functionality of the tools used for logic synthesis and place-and-route.

It is assumed that you have good knowledge on the basics of VHDL coding. The purpose of this assignment is not to teach these basics, but to teach you to use them and deepen your skills. Because you are not familiar with the synthesis and place-and-route tools, instructions to perform those steps are provided (in separate document).

The book you may use as a reference for VHDL language is: Peter J. Ashenden, "The designer's guide to VHDL", 3rd ed. For microcontroller functionality, it is strongly suggested to familiarize yourself with: Patterson & Hennessy, "Computer organization & design. The hardware & software interface", Morgan Kaufmann, 1998, 3rd ed.

1.1 Understanding the controller functionality

Get well acquainted with the datasheet of the PIC16F84A controller, try to understand its functionality and the instruction set. This is the most difficult part of the assignment, since you really have to understand the functionality of the controller before you can implement it.

From the implementation point of view, it is important to divide the execution of an instruction into phases:

1. Fetch instruction from program memory and decode it.
2. Fetch operand from memory (if needed).
3. Perform arithmetic (ALU).
4. Write memory and update registers.

Depending on the instruction, some of the phases can be skipped.

In general, you need a rising clock edge to write to a memory element. Memory block has data bus, address bus, read bus and write-enable signal. If write-enable is high, next rising edge of the clock will write the value of the data bus to memory location pointed by the address bus. To alleviate the design process, special-purpose registers (program counter, status register, etc.) may have dedicated data buses, but they should be accessible also through normal memory addressing.

In order to simplify the task, following exceptions are allowed.

1. Implementation of interrupt handling is not required. That means, you do not have to consider anything related to watchdog timer, power-up timer, start-up timer, or Timer 0 module. As a consequence, the following instructions can be discarded: CLRWDT, RETFIE, SLEEP.

2. The branch instructions (i.e. CALL, GOTO, conditional skips, etc.) do not need to be implemented. As a consequence, the 8-level stack can be left out, since its only function is to store the return addresses from subroutine calls. In addition, direct memory write to the program counter is not allowed (this would be equivalent to a branch).

3. If you need reset signal or multiple resets, implement them as independent external signals controlled by the test bench. Timing can be generated within the test bench with one clock signal.

4. Implementation of input/output ports can be reduced to corresponding memory addresses (PORTA and PORTB) directly connected to output ports.

Otherwise, your microcontroller should be able to execute the full instruction set as it is described in the datasheet. Please also refer to the intro lecture for details of what should/shouldn't be implemented.

1.2 Programming the controller

In order to get to know what happens in the controller, play around with the assembly design environment Piklab. The intention is that you could write simple programs in assembly language and run them with your microcontroller. Once you have the hexfile compiled from assembly code, you may read this hexfile to "program memory" inside your VHDL test bench with the provided hexfile reader function.

Here are some instructions to develop assembly programs for the controller.

- In course template package you have a "piklab" directory in which you will play with microcontroller model.

- In directory "piklab" there is assembly source "led_blinker.asm", which you can use as a simple example code.

- Go to "piklab" directory and start the design environment with command `piklab`.

- From “Project” menu select “New project”. Name can be for example “Led-blinker”, and the directory is the directory of the asm source code. Toolchain is “GPutils” and the device 16F84. Programmer is GPsim. (NOTE: The 16F84A version is not supported by GPsim).
- In source file selection, select “Add existing files”, then add “led_blinker.asm” to the project.
- You may view and edit the source file from the sources section.
- From “Build” menu, you may select “Compile file”. Compilation should be succesfull.
- From “Project” menu, select “Project options”. Under “Toolchain” tab there is “Linker” tab. Select hex file format “inhx8m”. This should be compliant with the VHDL hex code reader provided to you.
- From “Build” menu, select “Build project”.
- You may start the GPsim simulator from command line as `gpsim -p 16F84`. From field menu, select the appropriate .cod file.
- From “Windows” menu of GPsim select at least “RAM” and “Program memory”.
- Set the “Simulation mode” from the main window to be “Update GUI every cycle”.
- Try a couple of steps, to see how the program is executed and the register values are updated. You may also run for longer time in order to see how the counters are updated according to the program.

For testing your VHDL project, you may write any kind of assembly program, compile and build it in Piklab, and simulate it with GPsim. Here are some links that you might find helpful:
http://www.mikroe.com/en/books/picbook/0_Uvod.htm
http://www.winpicprog.co.uk/pic_tutorial.htm
http://www.hobbyprojects.com/microcontroller_tutorials.html
http://www.piclist.com/techref/member/jwn-hotmail-f41/TIMER_TUTORIAL.htm
And surely you may search for more, or develop as complicated code as you like. Nevertheless, remember that the main emphasis of this course is in microcontroller implementation, not programming it.

If you need any help, do not hesitate to ask.

2 How to pass

2.1 Returning the assignment

The study diary must be a PDF-file created with the provided Latex template. Reports violating the style of the template will not be graded. The study diary, VHDL code and any supplementary material (e.g., assembly program used for testing) must be packaged into a single .tar.gz file, and submitted via the provided return box in MyCourses. Please include in the package also a README file with instructions on how to demonstrate the functionality of the code, i.e., instructions for a single top-level test bench simulation. Deadline is 31st of May.

2.2 Evaluation

Course will be graded based on study diary and quality of the VHDL code. Study diary should document and describe the structure of the microcontroller, the phases of the design flow, difficulties encountered and how they were solved. The number of details included is up to the student. It is strongly suggested that the diary is truly written as a diary: every day you work on the project, you should write a paragraph or two about what you did, the problems you were trying to solve, and the solutions you have figured out.

2.3 Grading

In order to complete the course, the following phases of the IC design flow should be completed and documented.

- VHDL description of the PIC16F84A microcontroller.
- Simulation and verification of functionality.
- Logic synthesis and timing verification.
- Static functionality verification.
- Place-and-route.
- Final verification for timing and functionality.

Study diary and VHDL code will be evaluated based on following criteria.

1. Does the study diary reflect own thinking? Is the language technically sound? 1-5p

    - Very brief/unclear explanation what has been done, no reasoning. 1p
    - Short but clear description of the structure. 2p
    - More detailed explanation what has been done, but no reasoning why. 3p
    - Like above but contains also some indications of thinking behind decisions. 4p
    - Explanation what has been done and why. Reported decisions and reasons for them. 5p

2. Are the phases of digital IC implementation described in a way that reflects genuine understanding on what happens in the flow? 1-5p

    - Phases listed. 1p
    - Phases listed and their purpose explained. 3p
    - Detailed description of methods, constraints and scripts used, verifications performed, constraint related problems (if any) and solutions applied. 5p

3. Are the encountered problems and solutions, especially modifications required to make the VHDL synthesizable, documented thoroughly? 1-5p

    - Modifications listed. 1p
    - Causes of the problems and required modifications listed. 3p
    - Causes of the problems and required modifications explained in detail. 5p

4. Is the VHDL code clearly written and commented? 1-5p

    - Unclear structure, no/very little commenting, no proper indentation. 1p
    - Readable, commented, but not well structured (e.g. multiple entities or packages in one file). 3p
    - Readable and well commented/documented, organized in proper hierarchy with one entity or package per file. 5p

5. Does the structure of the code reflect understanding of the possibilities provided by the language (packages, functions, procedures)? Is the code efficiently structured? 1-5p

- Usage of components, processes, procedures, type definitions, signals, variables or functions reflects lack of understanding how these can be exploited to provide clear and modular code. 1p
- Components and processes used and well structured. Functions and procedures or packages not intentionally used for improved modularity. 3p
- Procedures, functions, types and packages used for improved modularity. 5p

6. Are there other positive indicators to be taken into account? 1-5p

The final grade will depend on the sum of the fields above according to the following table.

Points	Grade
12-13	1
14-16	2
17-19	3
20-22	4
23-	5