Problems on digital circuits and systems (CSD)

Francesc J. Robert
Josep Jordana

2020 (V3)
# Contents

Preface ................................................................................................................................. 9

Combinational circuits ........................................................................................................... 13

P1 Logic gates and Boolean algebra ......................................................................................... 14

Objectives ............................................................................................................................... 14
  1.1 Circuit analysis, truth tables and maxterms and minterms ............................................ 15
  1.2 Design Circuit_C using minimised equations ............................................................ 18
  1.3 Design Circuit_K using minimised equations ............................................................ 19
  1.4 Circuits using only NOR or only NAND ..................................................................... 20
  1.5 Circuits using only NOR or only NAND ..................................................................... 21
  1.6 Logic equations (PoS, SoP, maxterms, minterms) ...................................................... 22

P2 Standard logic circuits and flat VHDL design ................................................................. 23

Objectives ............................................................................................................................... 23
  2.1 Using VHDL EDA tools for synthesis and simulation .................................................. 24
  2.2 Designing a MUX_8 using several architectures ....................................................... 25
  2.3 Designing a HEX_7SEG_decoder ................................................................................ 26
  2.4 Designing a 10-to-4 line priority encoder .................................................................... 27
  2.5 Designing an 8-input priority encoder ......................................................................... 30
  2.6 A digital wind direction meter .................................................................................... 32

P3 Arithmetic circuits: adders, multipliers, comparators, etc. and VHDL
hierarchical design (plan C2) ................................................................................................ 37

Objectives ............................................................................................................................... 37
  3.1 Logic functions using the method of decoders ............................................................ 38
  3.2 Logic functions using the method of multiplexers ....................................................... 39
  3.3 Design a 1-bit full adder (flat) .................................................................................... 40
  3.4 Design a 1-bit full adder (structural) .......................................................................... 41
  3.5 Design a 1-bit comparator .......................................................................................... 42
  3.6 Designing a MUX_8 using a multiple-file structure ................................................... 43
  3.7 4-bit ripple adder ........................................................................................................ 44
  3.8 8-bit binary adder using 4-bit carry-look ahead adders .............................................. 45
  3.9 Designing a 6-bit comparator using VHDL ................................................................. 46
  3.10 Counting occupied parking slots (32-bit ones’ counter) ............................................ 47
  3.11 1-bit subtractor ......................................................................................................... 50
P4 Arithmetic circuits for 2C integer numbers and gate-level simulations for propagation delay measurements................................................................. 51

Objectives.................................................................................................................. 51
4.1 Addition and subtraction in two’s complement ..................................................... 52
4.2 Designing an 8-bit adder/subtractor for integer numbers .............................. 53
4.3 Designing a 10-bit comparator for radix-2 and 2C numbers ....................... 54
4.4 Performing gate-level simulations and propagation time measurements ................................................................. 55
4.5 How to design an 8-bit multiplier for 2C integer numbers? ........................ 56

Sequential systems .................................................................................................. 59

P5 1-bit memory cells: latches and flip-flops .......................................................... 60

Objectives.................................................................................................................. 60
5.1 Designing and using an RS latch. Deducing an RS_FF................................. 61
5.2 Data flip-flop (D_FF) ..................................................................................... 62
5.3 Analysis of a synchronous circuit .................................................................. 63
5.4 JK_FF and analysis of an asynchronous circuit............................................... 64
5.5 Analysis of a synchronous circuit .................................................................. 67
5.6 Design a toggle flip-flop (T_FF) using the FSM strategy ............................ 68
5.7 Design a JK flip-flop using the FSM strategy ................................................ 69
5.8 Analysis of an asynchronous counter (type 7493) ....................................... 70
5.9 Analysis of an asynchronous circuit based on T_FF .................................... 71
5.10 Design a combinational circuit using the method of ROM memories .......... 72
5.11 Design a HEX_7seg using the method of ROM memories .......................... 73

P6 Finite State Machines (FSM) ............................................................................. 74

Objectives.................................................................................................................. 74
6.1 Controlling the classroom luminaires............................................................... 75
6.2 Invent a bicycle torch ..................................................................................... 76
6.3 Debouncing circuit......................................................................................... 76
6.4 16-key matrix encoder................................................................................... 77
6.5 Water tank controller .................................................................................... 78
6.6 Traffic light controller .................................................................................. 80
6.7 Stepper motor controller ............................................................................... 81
6.8 7-segment digit sequencer ........................................................................... 82

P7 Standard counters and registers ...................................................................... 84

Objectives.................................................................................................................. 84
7.1 1-digit BCD counter ...................................................................................... 85
7.2 Synchronous universal 4-bit binary counter .................................................. 86
7.3 Synchronous modulo 12 counter ................................................................... 88
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.4</td>
<td>Data register</td>
<td>89</td>
</tr>
<tr>
<td>7.5</td>
<td>Shift register</td>
<td>90</td>
</tr>
<tr>
<td>7.6</td>
<td>Hour counter for a real-time clock</td>
<td>91</td>
</tr>
<tr>
<td>7.7</td>
<td>6-bit binary universal counter</td>
<td>93</td>
</tr>
<tr>
<td>7.8</td>
<td>Designing a Johnson counter</td>
<td>95</td>
</tr>
<tr>
<td>P8</td>
<td>Dedicated processors and advanced circuits</td>
<td>97</td>
</tr>
<tr>
<td>8.1</td>
<td>Generation of CLK signals</td>
<td>98</td>
</tr>
<tr>
<td>8.2</td>
<td>Pulse generator</td>
<td>99</td>
</tr>
<tr>
<td>8.3</td>
<td>Designing an industrial application</td>
<td>101</td>
</tr>
<tr>
<td>8.4</td>
<td>Design a 2-digit even/odd counter with start/stop button</td>
<td>103</td>
</tr>
<tr>
<td>8.5</td>
<td>Synchronous serial adder</td>
<td>103</td>
</tr>
<tr>
<td>8.6</td>
<td>Timer MMSS</td>
<td>104</td>
</tr>
<tr>
<td>8.7</td>
<td>Synchronous serial multiplier</td>
<td>104</td>
</tr>
<tr>
<td>8.8</td>
<td>Serial transmitter and receiver (USART)</td>
<td>104</td>
</tr>
<tr>
<td>8.9</td>
<td>Stepping motor control based on a dedicated processor</td>
<td>105</td>
</tr>
<tr>
<td>9.1</td>
<td>The microcontroller PIC16F</td>
<td>108</td>
</tr>
<tr>
<td>9.2</td>
<td>Invent a Dual_MUX4 based on a µC</td>
<td>111</td>
</tr>
<tr>
<td>9.3</td>
<td>1-digit BCD adder</td>
<td>112</td>
</tr>
<tr>
<td>9.4</td>
<td>12-to-4 encoder</td>
<td>114</td>
</tr>
<tr>
<td>P10</td>
<td>Programing FSM in C style. Events detection using interrupts</td>
<td>115</td>
</tr>
<tr>
<td>10.1</td>
<td>1-digit BCD counter</td>
<td>117</td>
</tr>
<tr>
<td>10.2</td>
<td>Binary counter modulo 256</td>
<td>118</td>
</tr>
<tr>
<td>10.3</td>
<td>4-bit serial data transmitter</td>
<td>119</td>
</tr>
<tr>
<td>10.4</td>
<td>5-bit Johnson counter</td>
<td>121</td>
</tr>
<tr>
<td>10.5</td>
<td>Stepper motor controller</td>
<td>122</td>
</tr>
<tr>
<td>P11</td>
<td>Peripherals: LCD display</td>
<td>123</td>
</tr>
<tr>
<td>11.1</td>
<td>Basic interface for a LCD display</td>
<td>123</td>
</tr>
<tr>
<td>11.2</td>
<td>Interfacing an I2C display</td>
<td>123</td>
</tr>
<tr>
<td>P12</td>
<td>Peripherals and more complex applications</td>
<td>124</td>
</tr>
<tr>
<td>12.1</td>
<td>Industrial application</td>
<td>125</td>
</tr>
<tr>
<td>Section</td>
<td>Title</td>
<td>Page</td>
</tr>
<tr>
<td>---------</td>
<td>-----------------------------------------------------------------------</td>
<td>------</td>
</tr>
<tr>
<td>12.2</td>
<td>Simple remote control</td>
<td>126</td>
</tr>
<tr>
<td>12.3</td>
<td>Non-retriggerable timer</td>
<td>128</td>
</tr>
<tr>
<td>12.4</td>
<td>Timers. PWM generation</td>
<td>130</td>
</tr>
<tr>
<td>12.5</td>
<td>Temperature measurement using timers</td>
<td>131</td>
</tr>
<tr>
<td>12.6</td>
<td>Temperature measurement using A/D converters</td>
<td>131</td>
</tr>
</tbody>
</table>

**Bibliography and internet links** .......................... 135

- **Bibliography** ........................................ 135
- **Internet links** ........................................ 135

**Examples of questions** .................................... 137

- **P1** .................................................. 137
- **P2** .................................................. 137

**Index** .................................................................. 139
This publication is the initial draft of a collection of problems and exercises from the former Digital Electronics (ED) and Digital Electronic Systems (SED) subjects and from past editions of the Digital Circuits and Systems (CSD) course for which this learning resource has been created. The publication, which is now under construction, will contain reviewed versions of design exercises from the three chapters in which CSD is organised: combinational circuits, finite state machines (FSM) and dedicated processors, and microcontrollers.

The aim of this publication is to help students to develop the following telecommunications engineering competencies associated with the CSD course:

- **CE 14 TELECOM.** A capacity for the analysis and design of synchronous and asynchronous combinatorial and sequential circuits, and the ability to use microprocessors and integrated circuits.
- **CE 15 TELECOM.** Knowledge of and the ability to apply the fundamentals of hardware description languages. (CIN/352/2009, BOE 20/2/2009)
- **PROJECT MANAGEMENT - Level 1:** The ability to use project management tools to carry out the stages of a project set by the professor.
- **EFFICIENT USE OF EQUIPMENT AND INSTRUMENTS - Level 1:** The ability to use the instruments, equipment and software of the laboratories for general or basic use and to carry out experiments and practicals and analyse the results.
- **INDEPENDENT LEARNING - Level 1:** The ability to complete tasks in the time allotted, using the suggested materials and following the guidelines set by the professor.
- **EFFECTIVE ORAL AND WRITTEN COMMUNICATION - Level 1:** The ability to complete tasks in the time allotted, using the suggested materials and following the guidelines set by the professor. The ability
to plan oral presentations, correctly reply to questions and write basic texts without spelling and grammar mistakes.

- **TEAMWORK - Level 1.** The ability to take an active role in group work, which includes identifying specific goals, determining collective and individual responsibilities and reaching a consensus on the most suitable approach to adopt for each problem.

- **FOREIGN LANGUAGE.** Knowledge of a foreign language, preferably English, at an oral and written level that is consistent with what is required of students on each degree.

We would appreciate your comments on this list of projects, so that we can enhance the process of finding errors and making improvements.

The table of contents is structured following the course web page.

---

**Course organisation and basic learning goals**


- Use your official UPC email address to communicate with your instructors for these reasons [9].

- Use and manage an e-mail client like Thunderbird [10] or Outlook.

- Use a SMB disk like your mapped “L” to carry out projects and assignments on school premises.

- Discuss the five elements required to achieve effective cooperative learning: positive interdependence; face-to-face interaction, individual
accountability and personal responsibility, use of interpersonal and small-group skills and group processing or reflection.

- Analyse and manage your individual and group study time. Be aware that 6 ECTS are equivalent to 150 hours of study time.
- Produce quality written solutions for your projects using pen-and-paper. Then (optionally) use this template to complete and save the solutions in electronic format. Generally, project solutions consist of specifications, plan, development, test, report and, in some selected exercises, prototyping.
- (Optional) Use Google sites or a similar application to build your cooperative group ePortfolio (with this template) and publish your projects, results and reflection.
- Assess your own or the group’s learning progression and the quality of the deliverables using the given rubric.
Combinational circuits
P1 Logic gates and Boolean algebra

Objectives
After studying the content of these projects, you will be able to:

- Use and explain the functionality of logic gates AND, NAND, OR, NOR, XOR, NXOR and NOT.
- Find datasheets of small and medium scale of integration (SSI and MSI) integrated circuits.
- Analyse a logic circuit built using logic gates (deduce its truth table).
- Explain and relate the following concepts for designing a logic circuit: truth table, canonical algebraic equations: minterms and maxterms, Boolean algebra and logic functions, minimisation: SoP (sum of products) and PoS (product of sums).
- Simplify or minimise logic functions using software like Minilog.exe.
- Use the application WolframAlpha to verify logic equations and determine the truth table of a combinational circuit.
- Use the HADES JAVA-based platform or Deeds to visualise and analyse the operation of digital circuits.
- Capture and simulate a schematic using the virtual laboratory software Proteus-ISIS or MultiSim.
- Search books and the internet to find information on the basics of VHDL language and explain the differences between VHDL design styles: structural and behavioural.
- Use the register transfer level (RTL) and technology schematic views to inspect the results of the synthesis process.
- Explain the basic technological details of an sPLD (22V10), CPLD or FPGA and how to program them to implement logic functions.
- Install computer-aided design (CAD) and electronic design automation (EDA) tools (Lattice Semiconductor ispLEVER Classic or Diamond, Intel Quartus II or Prime, and Xilinx ISE or Vivado), and run its design flow to implement VHDL projects for sPLD/CPLD/FPGA chips. Essentially the process involves VHDL source files, synthesis, functional simulation, pin assignment, gate-level simulation, target device programming and prototype verification.
- Simulate a logic circuit using EDA tools: ActiveHDL Lattice edition, ModelSim Intel edition or Xilinx ISim.
- Use sPLD/CPLD/FPGA development boards to prototype and verify the course projects.
1.1 Circuit analysis, truth tables and maxterms and minterms

The aim of this exercise is firstly to analyse circuits A and B in Fig. 1 to obtain their truth tables \( P = f(S1, S0, A, B) \); \( Q = f(S1, S0, A, B) \) and secondly to draw another equivalent circuit using the canonical logic equations (maxterms and minterms).

Let us establish a plan to solve this problem. Consider Circuit A and Circuit B as complete independent problems. First, solve Circuit A before continuing with
Circuit B. Fig. 2 shows several ways to plan how to determine the truth table of a given simple combinational circuit composed of logic gates.

**Phase A: Deduce and verify the circuit’s truth table:**

1. **Method I.** Draw the Circuit A, capture the schematics in Proteus and run simulations to obtain its truth table. There are up to 16 input combinations to complete the circuit truth table.

2. **Method II.** Deduce the logic equation that exactly matches the circuit. The numerical engine WolframAlpha can be used to obtain the truth table by typing the equation directly and analysing the computer results.

3. **Method III.** Apply Boolean algebra to determine the truth table (which is equivalent to the sum of minterms or the product of maxterms). In this way, the SoP or PoS expressions will be obtained as a step towards the final truth table.

4. **Method IV.** Run a VHDL design flow using EDA tools (a single-file structural project, circuit synthesis and test bench functional simulation) to produce a circuit. Verify the circuit by means of a timing diagram from which to annotate a truth table that must be identical to that deduced using any of the three previous methods.
**Phase B**: Invent several circuits from the given truth table, as shown in the map in Fig. 3.

5. Invent a canonical Circuit_3 using the product of maxterms.
6. From Circuit_3, obtain a new Circuit_5 based on 2-input NOR gates.
7. Create a new Circuit_2 by minimising the truth table and choosing PoS.
8. From Circuit_2, derive a new Circuit_4 based on 2-input NAND gates.

* Some ideas on the solution to the analysis problem can be found in this [tutorial](#).
1.2 Design Circuit_C using minimised equations

This exercise has two aims: first to analyse Circuit_C in Fig. 4 to obtain its truth table, and second to draw an equivalent circuit using minimised equations like the sum of products (SoP) or the product of sums (PoS).

Let us follow the plan depicted in Fig. 2:
- Phase A: obtain the circuit’s truth table by means of at least two of the four methods proposed, so that you can check that the truth table is correct.
- Phase B: invent another circuit using minimised equations.

Use an application like Minilog or Logic Friday to obtain minimised equations from a given truth table.

Some ideas on the solution to the analysis problem can be found in this tutorial.
1.3 Design Circuit_K using minimised equations

This exercise has two aims, first to analyse Circuit_K in Fig. 5 to obtain its truth table, and second to draw an equivalent circuit using minimised equations like the sum of products (SoP) or the product of sums (PoS).

Let us follow the plan shown in Fig. 2:
- Phase A: obtain the circuit’s truth table by means of at least two of the four proposed methods, so that you can check the truth table is correct.
- Phase B: invent another circuit using minimised equations.

Use an application like Minilog or Logic Friday to obtain minimised equations from a given truth table.

Some ideas on the solution of this analysis problem can be found in this tutorial.
1.4 Circuits using only NOR or only NAND

Given the following Boolean expression:

\[ Q = f(x, y, z) = x \cdot y + y' \cdot z \]

a) Draw the circuit’s truth table and symbol. Represent the circuit using only NAND logic gates.

b) Express the output as a sum of minterms and a product of maxterms.

c) With the equation obtained in b), represent the circuit using only-NOR logic gates.

d) Calculate the maximum frequency of operation of this circuit if the propagation delay time \( t_{\text{PHL}} \) and \( t_{\text{PLH}} \) of a gate of this kind, for instance 74HCT technology is 21 ns.

e) How would you use the VHDL simulator to verify the truth table of this digital circuit?

The representation in Fig. 6 is a map that can help to comprehend the concepts. Once you have deduced the circuit’s truth table, you can produce several circuits that meet the same specifications.
1.5 Circuits using only NOR or only NAND

Interpret the following table output format in Fig. 7 from Minilog.
- Draw the symbol of the circuit.
- First draw the logic circuit for the output \(A_L = f_1(D, C, B, A)\) using only NOR gates, and second modify the circuit so that all the gates are 2-input NOR.
- First draw the circuit for the output \(B_L = f_2(D, C, B, A)\) using only NAND gates, and second modify the circuit so that all the gates are 2-input NAND.
- How many maxterms and minterms contain the function \(E_L = f_3(D, C, B, A)\)?

**MINIMISATION RESULT STATISTICS**

<table>
<thead>
<tr>
<th></th>
<th>100</th>
<th>110</th>
<th>011</th>
<th>111</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>DCBA</td>
<td>LLL</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ABE</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAXIMUM FANIN:</td>
<td>18</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOTAL LITERAL COUNT:</td>
<td>102</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAXIMUM FACTOR SIZE:</td>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MAXIMUM OUTPUT FUNCTION SIZE:</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Remember that you must deduce the equivalent equations before you draw the circuits.

Fig. 7
Output table format from a given circuit described in Minilog.
1.6 Logic equations (PoS, SoP, maxterms, minterms)

Fig. 8 shows the block diagram and the truth table of an 8-to-3 encoder (Enc_8_3), a typical next section example of standard circuit. When several inputs are active high at the same time, a binary code is generated of the highest priority signal. The symbol “-” means a “don’t care” value that is represented other times by “x”. GS goes high when any input is assessed, thus it can be used both as a flag to indicate that a key is pressed and for disambiguation of the code “000”.

- Represent the output $Y_2 = f(X_7 \ldots X_0)$ using a product of sums (PoS).
- Represent the output $Y_1 = f(X_7 \ldots X_0)$ using a sum of products (SoP).
- Represent the output GS using maxterms. How many minterms will the function have?
- Write down the single-file (flat) VHDL code using either structural (plan A) or behavioural (plan B) style. Explain the differences between the two coding styles.
P2 Standard logic circuits and flat VHDL design

Objectives

After studying the content of these projects, you will be able to:

- Explain the specifications and characteristics of the standard combinational logic blocks: multiplexers (or data selectors), demultiplexers (or data distributors), decoders, encoders, hexadecimal to seven-segment LED displays adapters, code converters, etc. Specifications include concepts like: symbol, truth table and functionality, internal design, expandability, and commercial chips of similar characteristics.
- Explain the functionality of the enable input that is available in most of these standard circuits.
- Explain the concepts of flat and hierarchical designs and implement simple projects in VHDL involving a single file (flat) using structural (plan A) or behavioural (plan B) approaches.
- Explain how to chain or expand such devices to implement a larger one, for instance, how to connect several MUX_4 to obtain a MUX_16.
- Implement standard circuits targeted at a given PLD (CPLD or FPGA) using VHDL and synthesis and simulation EDA tools. Explain the VHDL design flow.
- Find datasheets of classic logic circuits from different technologies.
- Explain how to interface switches to encoders.
- Explain how to interface LEDs and seven-segment display to decoder outputs.
2.1 Using VHDL EDA tools for synthesis and simulation

Calculate the truth table of the circuit depicted in Fig. 9 using VHDL EDA tools. This is the analysis method #4 presented in P1 and a way to discover the concepts associated with the VHDL design flow.

a) Understand the specifications and characteristics of VHDL synthesis and simulation tools.

b) Organise a plan detailing the sequence of operations to reach the end of the problem successfully.

c) Develop the solution.

d) Test the truth table using other methods of circuit analysis such as those described in P1.

Problem discussion
2.2 Designing a MUX_8 using several architectures

The objective is to design the functionality of a MUX_8 type 74HCT151 using VHDL synthesis and simulation EDA tools. The circuit MUX_8 is simple and so it must be designed flat, which means a single VHDL file\(^1\) is used to describe the complete architecture. Three plans are presented, so this problem is completely divided into three projects. Plan, develop and test them separately. This reference is a link to the VHDL design flow that states the entire sequence of operations required to succeed in the design.

- Plan A. Structural, using logic equations.
- Plan B. Behavioural, using the truth table or a high-level algorithm describing the chips functionality.
- Plan C1. Invent a hierarchical schematic for MUX_8 using smaller blocks of the same kind, for instance, MUX_4 or MUX_2.

\[\text{Problem discussion}\]

---

\(^1\) The plan C2 proposed in P3 can be used to design hierarchical structures using multiple-file projects, which enables the design of complex large circuits.
2.3 Designing a HEX_7SEG_decoder

The objective is to design the functionality of a HEX_7SEG_decoder type 74LS47 using VHDL synthesis and simulation EDA tools. The symbol and truth table adapted from the datasheet are represented in Fig. 11. The design must be flat with all the architecture included in a single VHDL file. Two plans are presented, so this problem is divided completely into two projects. Plan, develop and test them separately. This reference is a link to the VHDL design flow that states the entire sequence of operations required to succeed in the design.

- Plan A. Structural, using logic equations.
- Plan B. Behavioural, using the truth table or a high-level algorithm describing the chip’s functionality.

Problem discussion
2.4 Designing a 10-to-4 line priority encoder

The objective is to develop VHDL code and the final circuit for the 10-to-4 encoder circuit (Enc_10_4) component represented in Fig. 12, which can be interfaced to a standard 10-key numeric keypad. The circuit must have high priority in case several keys are pressed at the same time. This circuit is similar to a standard combinational chip like the 74LS148. As usual, we can plan the project in several ways as represented in our CSD design flow chart:
- Plan A: Structural (flat design with a single VHDL file), using the truth table logic equations in a canonical or simplified version.
- Plan B: Behavioural (flat design with a single VHDL file), using the high-level description of the specifications.
- Plan C2: Structural (hierarchical design with multiple VHDL files), building the project using an architecture consisting of smaller components of the same kind, such as Enc_4_2 or Enc_8_3.

As usual in these problems, you will solve some drilling exercises on Boole’s algebra before you attempt the design of the Enc_10_4.
Section 1: Specifications and theory

1) Find on the internet a commercial standard circuit in classic technologies which has a similar truth table. You can start by visiting our list.

2) Fill in the truth table using the names and the variable order depicted below using “don’t care” (“-” or “x”) terms. \( G_s \) has to be high (‘1’) when any key is pressed. The enabled output (\( E_o \)) is high when the circuit is enabled (\( E_i = ‘1’ \)) and the keys are not pressed. How many combinations does the truth table include?

| \( E_i \) | \( K_9 \) | \( K_8 \) | \( K_7 \) | \( K_6 \) | \( K_5 \) | \( K_4 \) | \( K_3 \) | \( K_2 \) | \( K_1 \) | \( K_0 \) | \( D_3 \) | \( D_2 \) | \( D_1 \) | \( D_0 \) | \( G_s \) | \( E_o \) | Comments |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
|       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |       |

3) Draw the sketch of a timing diagram for the circuit and describe the outputs that are expected when different inputs are applied.

4) How long does it take to run a complete full verification of the circuit if each input vector has a duration of \( \text{Min\_Pulse} = 10 \mu s \)?

5) How to generate a ‘0’ or a ‘1’ using buttons or switches? How do you drive a LED, for example to connect at the output \( G_s \)? Study this circuit in Proteus to get ideas and design formulas.

6) How many minterms will \( D_3 \) have? How many maxterms will \( D_2 \) have?

7) Inspect the truth table or use Minilog to represent the six output functions either as the sum of products (SoP) or the product of sums (PoS).

8) Draw the logic circuit of the output \( D_1 \) using only NOR gates.

\[
(x \cdot y)' = x' + y' \\
(x \cdot y \cdot z)'' = (x' + y' + z')'
\]

9) If the propagation delay of a single gate of the technology used is 7.5 ns, what is the maximum frequency of operation of the circuit in 8)?

10) Prepare other similar questions such as: invent \( D_0 \) using only NAND, write the text file in tbl format for the Minilog minimiser, etc.

Section 2: Select a plan and complete the project

The following questions are related to synthesising the project \( \text{Enc\_10\_4} \) into a programmable device and testing it using EDA tools. The plan to organise the architecture is either 11), 12) or 13). Each plan leads to a different project.

11) Architecture #1 (Plan A, structural-flat): write down the structural VHDL code that is derived from the equations deduced in 7).
12) Architecture #2 (Plan B, behavioural): search the internet, digsys web or find in books a high-level or algorithmic VHDL code for the component in Fig. 12b. As usual, use flow charts or schematics to translate the truth table into VHDL.

13) Architecture #3 (Plan C2, structural-hierarchical): figure out how to design an Enc_10_4 using components like Enc_4_2 and other circuits if necessary. How many VHDL files will the project contain? In this section, the Ei and Eo signals must be used to facilitate block expansion.

Section 3: Develop your plan

14) Create a project for a CPLD or FPGA target chip, using EDA tools (Lattice ispLEVER Classic or Diamond, Intel Quartus Prime or Xilinx ISE or Vivado). Print the RTL schematic and technology schematics and discuss them. Can you identify the look-up tables (LUT), macrocells or logic elements used in the implementation?

Section 4: Test your circuit

15) Translate the timing diagram sketch represented in 3) into a VHDL test bench file to simulate functionally the circuit using ActiveHDL, ModelSim or ISim. Print the logic analyser timing diagram and discuss it.

16) Perform a gate-level simulation to measure the worst-case propagation delay and calculate the encoder’s maximum frequency of operation for a given PLD target chip.

Looking forward: this project introduces standard combinational encoder devices. They can be compared to matrix implementations such as in Problem 6.4.
2.5 Designing an 8-input priority encoder

Fig. 13 shows the internal circuit of the classic chip HEF4532B, an 8-input priority encoder and its truth table as specified by Philips.

![Fig. 13](image)

1) Redraw the truth table using ‘0’ and ‘1’ and explain what the circuit’s function is, using a pair of examples. How many binary combinations does this table have? Because each manufacturer of this industry standard chip uses its own style to name inputs and outputs, rename the circuit’s ports using our own CSD naming conventions: $E_{\text{in}} \rightarrow E_i$, $E_{\text{out}} \rightarrow E_o$, $I(7..0) \rightarrow D(7..0)$, $O(2..0) \rightarrow Y(2..0)$.

2) Draw a sketch of a timing diagram.

3) Write $E_o = f(E_i, D_7, ..., D_0)$ using minterms. In addition, draw the equivalent circuit using gates.

4) Write $GS$ as a product of sums (PoS). How many minterms does this function have? Draw the circuit using gates.

Write $Y_2$ as a sum of products (SoP). How many maxterms does this function have? Write $Y_1$ and $Y_0$ as a product of sums (PoS).

5) Describe the circuit in VHDL in a behavioural (plan B) fashion.

---

2 [http://digsys.upc.es/csd/P02/P2.html](http://digsys.upc.es/csd/P02/P2.html)
6) Describe the circuit in VHDL in a structural (plan A) fashion.

7) Translate the timing diagram in 2) into a VHDL test bench so that the circuit can be verified using an EDA tool.

Optional questions related to further understanding the problem and synthesising the project `encoder_8_to_3.vhd` into a programmable logic device (sPLD, CPLD) or a field programmable gate array (FPGA):

8) Write the `.tbl` format file so that it can be used to obtain the PoS or the SoP in Minilog software.

9) Find and write down the link to a similar circuit in HADES\(^3\) that can be executed using the Java applet.

10) Simulate the circuit `Enc_8_3` in Proteus and check if it works as expected.

11) Start a VHDL-based synthesis project and a testbench-based simulation using EDA tools for target CPLD or FPGA programmable chips. You can compare structural (plan A) and behavioural (plan B) implementations.

Such combinational circuits are considered again in Chapter III for learning the basics on microcontrollers in project P9.

---

\(^3\) HADES: [https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/index.html](https://tams.informatik.uni-hamburg.de/applets/hades/webdemos/index.html)
2.6 A digital wind direction meter

We want to design a digital wind direction meter \(\text{wind\_compass}\) as shown in Fig. 14, based on a 16-position optoelectronic rotary encoder. As shown in Fig. 15, the sensor disk is coded in Gray, which was originally used instead of binary code to prevent spurious outputs from electromechanical switches. The objective is to develop the VHDL code and the final circuit to be synthesised into a complex programmable device (CPLD) or a field programmable gate array (FPGA) chip.

To promote class and cooperative group discussions, we can plan the project in several ways, as represented in our CSD design flow chart. Each plan means a different project and circuit realisation that is useful for comparing solutions:

- Plan A: Structural (flat design with a single VHDL file), using logic equations in a canonical or simplified version.
- Plan B: Behavioural (flat design with a single VHDL file), using high-level description of the specifications.
- Plan C2: Structural (hierarchical design with multiple VHDL files), building the project using an architecture consisting of components and signals.

Fig. 16 shows the symbol of the \(\text{wind\_compass}\) chip to be implemented.

Section 1: Specifications and theory

Let us solve some initial drilling questions to learn a bit of theory and clarify ideas, and implement the projects based on plans A and B.

1) Write the truth table of the \(\text{wind\_compass}\). The inputs have to be ordered in this way: \(\text{E, D}(3..0)\).
2) Write the functions \(Y(7)\) and \(Y(14)\) canonically as a product of maxterms.
3) Write the functions \(S(6)\) and \(S(1)\) canonically as a sum of minterms.
4) Let us minimise the wind_compass using Minilog and obtain the table output formats for SoP and PoS.

5) Write the functions $S(2)$ and $Y(13)$ as an SoP and draw the equivalent logic circuit.

6) Write the functions $S(0)$ and $Y(11)$ as a PoS and draw the equivalent logic circuit.

7) Write the functions $S(4)$ and $Y(5)$ using only NOR.

8) Write the functions $S(3)$ and $Y(10)$ using only NAND.

9) Draw a schematic to translate, according to plan B the wind_compass truth table to VHDL, representing the required signals to interface the truth table artefact.

Fig. 15
The basics of the Gray to binary rotary encoder sensor.

Fig. 16
Symbol of the project wind_compass representing and naming all the inputs and outputs. Anyone of the 23 outputs is a function of the inputs $E$, $D(3)$, $D(2)$, $D(1)$, $D(0)$ in this order, for instance:

$$S(2) = f(E, D)$$

Fig. 17 shows an example of an internal electronic schematic for the wind_compass when plan C2 is used. It is available [here](#) as a Proteus capture for experimentation. The output of Chip1 (gray_bin_converter) is connected to both Chip2, a 1-digit 7-segment decoder (hex_7seg_decoder), and Chip3, a 16-bit decoder (dec_4_16) with one-hot output to light a wheel of 16 LED to display the wind direction.
Fig. 17

Internal design for the project wind_compass based on plan C2. The picture shows Gray code “1101”, which is “1001” in binary and corresponds to the wind direction South-East “SE”. In the way it is connected, the code “0000” corresponds to the direction “NNW”, and it advances counter-clockwise up to the code “1111” which is the direction “N”.

10) Run the Proteus circuit of the circuit in Fig. 17. Print the screen results when you input the Gray code “0101” and explain how it works.

11) Draw an example timing diagram showing the input stimulus and output responses. Assuming that Min_Pulse = 1.26 us, how long does it take to simulate all the circuit specifications?
Section 2: Select a plan

12) Plan A: write the VHDL code for the wind_compass to obtain the wind_compass.vhd circuit file.
   \[L:\text{CSD}\text{P2}\text{wind_compassA}\text{wind_compass.vhd}\]

13) Plan B: write the VHDL code for the wind_compass to obtain the wind_compass.vhd circuit file.
   \[L:\text{CSD}\text{P2}\text{wind_compassB}\text{wind_compass.vhd}\]

Section 3: Develop your plan
14) Synthesise the projects for the given target CPLD/FPGA chip using an EDA from Lattice, Xilinx or Intel/Altera. Discuss using handwritten comments the RTL and the technology schematics for plans A and B.

Section 4: Test your circuit
15) Test both projects functionally using the same VHDL test bench, for instance, derived from the timing diagram in Question 11). Print the logic analyser timing diagrams and explain them using handwritten comments.

There is a former exam (1718Q1, Prob. 1) that includes a solution with comments and simulation files for a similar project.

Optional extra questions to prepare P3 and P4 projects on VHDL hierarchical design and timed gate-level simulations:

16) Plan C2: design the project wind_compass using a multiple-file hierarchical approach.
   \[L:\text{CSD}\text{P3}\text{wind_compass}\text{files}\]

17) Perform a gate-level simulation to measure the worst-case propagation delay and calculate the encoder’s maximum frequency of operation for a given PLD target chip.
Fig. 18 shows additional details to help you to analyse the project, like the truth table associated with a combinational circuit such as Chip1 in Fig. 17, a 4-bit gray_bin_converter.

Table Gray_Bin_Converter

<table>
<thead>
<tr>
<th>A3</th>
<th>A2</th>
<th>A1</th>
<th>A0</th>
<th>B3</th>
<th>B2</th>
<th>B1</th>
<th>B0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Table translation into Minilog text format:

```
input A3 A2 A1 A0
output B3 B2 B1 B0
" INPUTS  OUTPUTS
" ===========  ============
" A3 A2 A1 A0   B3 B2 B1 B0
" = ===========  = ============
 0  0  1  0  0  0  0  0
 0  0  1  0  0  0  0  1
 0  1  1  1  0  0  0  0
 0  1  1  1  0  0  0  1
 0  1  0  0  0  0  0  0
 0  1  1  0  1  0  0  0
 0  1  1  0  0  0  0  1
 0  1  1  0  0  0  1  0
 0  1  1  0  1  0  0  1
 0  1  1  0  1  0  0  1
 0  1  1  0  1  0  1  0
 0  1  1  0  1  0  1  1
 0  1  0  1  0  0  1  0
 0  1  0  1  0  0  1  1
 1  1  1  1  0  1  0  0
 1  1  1  1  1  0  0  1
 1  1  1  1  1  1  0  1
 1  1  1  1  1  1  1  1
```

Note how the 16 input combinations do not have to be written necessarily in binary sequential.
P3 Arithmetic circuits: adders, multipliers, comparators, etc. and VHDL hierarchical design (plan C2)

Objectives

After studying the content of these projects, you will be able to:

- Explain how to perform basic operations like add, compare or multiply using radix-2 binary number system.
- Convert natural (whole) numbers between several number systems such as binary (radix-2), decimal (radix-10) or hexadecimal (radix-16).
- Find and analyse the characteristics of classic industry standard arithmetic chips like 74HCT283, 74LS85, 74LS181, etc.
- Write an alphanumeric message using ASCII code.
- Encode data, information or symbols in Gray, one-hot, BCD, etc.
- Explain the basics and operability of adders and comparators.
- Infer the idea of an arithmetic and logic unit (ALU) circuit.
- Infer how to design hardware multipliers.
- Infer an $n$-bit ripple-carry adder.
- Infer a 4-bit carry-lookahead adder and be able to compare its characteristics with respect to the ripple-carry adder.
- Use the hierarchical method of decoders (MoD) to implement logic functions.
- Use the hierarchical method of multiplexers (MoM) to implement logic functions.
- Apply hierarchical structural design (plan C2) to implement arithmetic circuits using VHDL.
3.1 Logic functions using the method of decoders

The following function is expressed in SoP:

\[ f(w, x, y, z) = x'y'z + x'y'z' + wxy' + wyz' + xy \]

a) Draw the entity’s symbol and draw the circuit diagram using logic gates. Write the equation in VHDL.

b) Apply Boole algebra analysis or use a computer tool like WolframAlpha or Logic Friday to deduce the truth table and the canonical equations sum of minterms and product of maxterms.

c) Solve the circuit by the method of decoders (MoD), a hierarchical plan C2 based on components. How many VHDL files are required to implement the project of this circuit?

d) Invent a timing diagram to demonstrate how the circuit works and translate it into a VHDL test bench to perform an ActiveHDL/ModelSim/ISim functional simulation.

In this unit you can learn the basics of the MoD for implementing logic functions.
3.2 Logic functions using the method of multiplexers

The following function is expressed in SoP:

\[ f(w, x, y, z) = x'y'z + x'y'z' + wxy' + wyz' + xy \]

a) Draw the entity’s symbol and draw the circuit diagram using logic gates. Write the equation in VHDL.

b) Apply Boole algebra analysis or use a computer tool like WolframAlpha or Logic Friday to deduce the truth table and the canonical equations sum of minterms and product of maxterms.

c) Solve the circuit by the method of multiplexers (MoM) using a MUX4, a hierarchical plan C2 based on components. How many VHDL files are required to implement the project of this circuit?

d) Invent a timing diagram to demonstrate how the circuit works, and translate it into a VHDL test bench to perform an ActiveHDL/ModelSim/ISim functional simulation.

In this unit you can learn the basics of the MoM for implementing logic functions.

Note that other projects can be inferred when considering applying the MoM and another type of multiplexer, for instance:

e) Solve the circuit by the method of multiplexers (MoM) using a MUX2. How many VHDL files are required to implement the project of this circuit?

f) Solve the circuit by the method of multiplexers (MoM) using a MUX8.

g) Solve the circuit by the method of multiplexers (MoM) using a MUX16.
3.3 Design a 1-bit full adder (flat)

Study and run the tutorial on the design of the Adder_1bit following two different single-VHDL file (flat) plans:
   A) Structural
   B) Behavioural

Problem discussion

Solve the additional questions:
   a) Implement \( C_o \) using only NOR gates.
   b) Implement \( S_o \) using maxterms.
   c) If a logic gate has a propagation delay of 6.5 ns, deduce the maximum frequency of operation of your circuit.
3.4 **Design a 1-bit full adder (structural)**

Study and run the tutorial on the design of the Adder_1bit following two different structural multiple-VHDL file (hierarchical) plans:

A) Method of decoders
B) Method of multiplexers

Problem discussion using the method of decoders

Problem discussion using the method of multiplexers
3.5 Design a 1-bit comparator

Study and run the tutorial on the design of the Comp_1bit based on SoP equations.

+ Problem discussion
3.6 Designing a MUX_8 using a multiple-file structure

The objective is to design the functionality of a MUX_8 type 74HCT151 using VHDL synthesis and simulation EDA tools and a hierarchical strategy using multiple VHDL files.

- Plan C2. Invent a hierarchical schematic for the MUX_8 using smaller components of the same kind, for instance, MUX_4 or and MUX_2.

Fig. 20  MUX_8 chip symbol derived from the standard 74HCT151.
3.7 4-bit ripple adder

Follow and run the tutorial on the Adder_4bit based on a ripple carry plan.

+ Problem discussion
3.8 8-bit binary adder using 4-bit carry-lookahead adders

Project:

a) Write the VHDL code of the complete 4-bit carry lookahead adder (Adder_1bit.vhd, Adder_4bit.vhd, Carry_generator.vhd).

b) Start a project using an EDA tool and synthesise the circuit in a given target chip CPLD or FPGA. Print the RTL view and comment it.

test the 4-bit carry-lookahead adder.

c) Start a VHDL simulator EDA tool and run a test bench to verify the unit under test applying some 4-bit operands. Print the timing diagram and add notes and explanations.
3.9 Designing a 6-bit comparator using VHDL

The idea is to develop and implement an expandable Comp_6bit component. Fig. 22 shows the entity’s symbol and the proposed internal architecture for the Plan C2 consisting of a structural hierarchical planning.

![Diagram showing the entity's symbol and the proposed internal architecture for the Comp_6bit component.](image)

**Table 23**

<table>
<thead>
<tr>
<th>A[2..0]</th>
<th>B[2..0]</th>
<th>Gi</th>
<th>Ei</th>
<th>Li</th>
<th>GT</th>
<th>EQ</th>
<th>LT</th>
</tr>
</thead>
<tbody>
<tr>
<td>A &gt; B</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>A &lt; B</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>A = B</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Plan C2: Implementation of a structural design in a CPLD or a FPGA.

a) Explain the design flow you will follow to produce your circuit using Lattice ispLEVER Classic / Altera Quartus II / Xilinx ISE.

b) Draw a structured hierarchical design as in Fig. 22b using several components. For instance, Fig. 23 shows the truth table for a 3-bit cascadable comparator.

c) Implement the elemental Comp_1bit using the logic equations derived from Minilog.exe (single output mode, sum of products, table output format). Verify your equations using WolframAlpha. This section is solved as a tutorial in the web (Comp_1bit).

d) Create a multiple-file VHDL-based project using EDA tools for a CPLD target chip, for example Lattice ispMACH4128V TQFP100, or the Intel-Altera MAX EPM7128SLC84-7, or the Xilinx CoolRunner II XC2C256-TQ144-7. Print and comment the RTL and technology views of the synthesised circuit, so that it can be compared to the initial block diagram proposed in Fig. 22b.
e) Test and simulate your design using the ActiveHDL / ModelSim / ISim simulators by means of a VHDL test bench.
   (Optional)

f) Assign pins and generate the output configurations files if the circuit has to be prototyped in a development board (Lattice HWD-LC4128V, Altera UP2 or Xilinx CoolRunner-II CPLD Starter Board).

g) Write down a report to document your design using our quality standards and templates.
   (Optional)

There are other ways to describe the same circuit, which are not in the scope of this introductory CSD course. Sometimes, the single-file behavioural version of the arithmetic circuit is not that difficult to write.

Plan B: Implementation of a behavioural design (flat design, single VHDL file)

h) Draw the truth table and a timing diagram sketch for the Comp_6bit circuit.

i) Write down the high level or behavioural VHDL code directly as a single block as in Fig. 22a planning writing first an algorithm or a flowchart to translate the circuit’s truth table.

j) Create a single-file VHDL project using the EDA tools to synthesise a circuit for a simple programmable logic device (sPLD) GAL22V10 (24 pins) or a CPLD or a FPGA chip. Print and comment the RTL and the technology views of the synthesised circuits.

k) Test and simulate your design using Proteus and its EasyHDL scripting language (in case of a sPLD). In case of a target chip CPLD or FPGA, use ActiveHDL / ModelSim / ISim simulators by means of a VHDL test bench, thus, translating the timing diagram sketch into VHDL to apply input stimulus. Print the timing diagrams and comment them.

3.10 Counting occupied parking slots (32-bit ones’ counter)

This project aims to represent in 7-segment displays the number of occupied parking slots. Each slot has installed an ultrasonic presence sensor which gives a ‘1’ when occupied. Thus, the first idea here for planning the entity Parking_occupancy in Fig. 24 is to consider components such as a Ones_counter_32bit where for example an input vector such as \( D = “1001 0011 1110 1111 1000 1111 1010 1110” \) will produce an output \( K = (010101)_2 = (21)_{10} \); a Converter_bin_BCD_6bit where for example an input such as \( K = (010101)_2 \) will generate and output \( T = “0010” \), \( U = “0001” \); and a pair of HEX_7seg_decoder to drive the 7-segment displays.
Fig. 24 Example of a parking occupancy monitor to calculate the number of occupied parking slots. In this example it is represented the number 21, meaning this number of detected cars in any position in the parking. 

VCC = 5 V

a) Draw and explain the internal architecture of the parking occupancy circuit based on components and representing some examples of the components truth tables.

b) The HEX_7seg_decoder has active-low outputs to drive a common-anode display and its technology is LS-TTL with the characteristics represented in the table. Calculate the value of the limiting resistor \( R_1 \) in the worst case scenario if each segment must be biased with 15 mA when lighting.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Limits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Min</td>
</tr>
<tr>
<td>( V_{IH} )</td>
<td>Input HIGH Voltage</td>
<td>2.0</td>
</tr>
<tr>
<td>( V_{IL} )</td>
<td>Input LOW Voltage</td>
<td></td>
</tr>
<tr>
<td>( V_{OH} )</td>
<td>Output HIGH Voltage</td>
<td>2.7</td>
</tr>
<tr>
<td>( V_{OL} )</td>
<td>Output LOW Voltage</td>
<td>0.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Parameter</th>
<th>Min</th>
<th>Max</th>
<th>Unit</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t_{PLH} )</td>
<td>Propagation Delay Time LOW-to-HIGH</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
<tr>
<td>( t_{PLH} )</td>
<td>Propagation Delay Time LOW-to-HIGH</td>
<td></td>
<td></td>
<td>ns</td>
</tr>
</tbody>
</table>

VCC = 1.7 V

Vred

\( R_1 \)

VCC = 5 V

4)

The Converter_bin_BCD_6bit is used to translate 6-bit radix-2 numbers to 2 BCD digits. Assuming the circuit is based on equations PoS (plan A) and implemented in LS-TTL technology where each gate has propagation delays as indicated in the table, calculate the maximum speed of computing.

c) The Converter_bin_BCD_6bit is used to translate 6-bit radix-2 numbers to 2 BCD digits. Assuming the circuit is based on equations PoS (plan A) and implemented in LS-TTL technology where each gate has propagation delays as indicated in the table, calculate the maximum speed of computing.

d) Invent the architecture of the ones_counter_32bit as a hierarchy of components (plan C2). For instance, Fig. 25 represents the schematic of a ones_counter_8bit. How many VHDL files will include this project? Check that your circuit works applying some input vectors.
Problem discussion including the 8-bit and the 4-bit ones’ counter and a commented solution.
3.11 1-bit subtractor

We want to implement a circuit for subtracting 8-bit binary numbers as shown in Fig. 26.

\[ D = A - B \]
\[ 0 \leq A, B \leq 2^n - 1 \]
\[ A \geq B \]

The strategy is to use a chain of simple 1-bit subtractors instead of the typical 1-bit adders and the two's complement convention. Thus, the circuit will work only with positive integers. The Fig. 27 shows the building block.

So, we can chain many 1-bit subtractor connecting the “borrows” in the same way we connect the “carry” when adding:

\[
\begin{array}{c}
\times \times \times \\
1 1 1 0 1 1 1 0 \\
1 0 1 1 1 1 \\
-------------- \\
1 1 0 1 0 1 1 1
\end{array}
\]

The full 1-bit subtractor (SUB_1) have the following truth table: \( D = f(A, B, B_{in}) = \Sigma m(1, 2, 4, 7); B_{out} = g(A, B, B_{in}) = \Pi M(0, 4, 5, 6) \)

1. Representing the truth table by means of equations, implement the 1-bit subtractor using only NAND gates. Write the equations in VHDL.
2. Write the code for the 1-bit subtractor in VHDL using a behavioural approach.
3. Draw the schematic of the 8-bit ripple subtractor (SUB_8) and describe it in VHDL using components.
4. Implement the logic circuit of a pair of flags or indicators to detect special events like:
   - A zero result \( D = A - B = 0 \)
   - A negative number \( D < 0 \) (A<B)

5. Draw a sketch of a timing diagram and write it as a VHDL test bench to test your design. Try at least three operations:
   - \( A = 230, B = 45 \)
   - \( A = 187, B = 177 \)
   - \( A = 177, B = 187 \)
P4 Arithmetic circuits for 2C integer numbers and gate-level simulations for propagation delay measurements

Objectives

After studying the content of these projects, you will be able to:

- Use standard arithmetic blocks for integer numbers based on two’s complement (2C) convention: subtractors, adders, comparators, multipliers, etc.
- Solve arithmetic operations for integers in two’s complement format (2C).
- Explain the range of an N-bit integer number in 2C and the meaning of an overflow operation result.
- Design combinational circuits in a hierarchical way using multiple combinational circuits as components (plan C2).
- Explain why a XOR gate can be considered a programmable gate that can be both, an inverter or a buffer.
- Perform VHDL gate-level simulations to calculate propagation delays and the maximum speed of computation of a given circuit.
- Discuss the main features of the electronic technology behind a CPLD or FPGA.
- Implement circuits in target PLD chips (CPLD or FPGA) populating the laboratory training boards from Xilinx, Intel-Altera or Lattice Semiconductor.
- Design circuits that can operate on different types of data, for instance, natural and integer numbers.
4.1 Addition and subtraction in two’s complement

1. Draw the symbol and the internal schematic of a 6-bit two’s complement adder/subtractor and determine the range of the operants and the result. Explain how the overflow (OV) flag works.

2. Perform the following operations in binary using the two’s complement (2C) 6-bit adder/subtractor from previous section 1). Check the result and deduce the Z and OV flags.
   
   a) \((+26)_{10} + (101010)_{2C}\)
   b) \((101010)_{2C} - (-21)_{10}\)
   c) \((+18)_{10} + (101110)_{2C}\)
   d) \((-31)_{10} - (010110)_{2C}\)

3. Represent the previous operations in a timing diagram and translate it (only the stimulus section) to a VHDL test bench using a constant \(Min\_Pulse = 7.5 \mu s\).

4. Determine the maximum speed of operation of the 6-bit 2C adder/subtractor if synthesised in a Xilinx technology Coolrunner CX2C256 CPLD that has the propagation delays shown below. Justify your calculations.

<table>
<thead>
<tr>
<th>Symbol</th>
<th>XC2C256 CoolRunner-II CPLD Parameter</th>
<th>(-6)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(T_{PD1})</td>
<td>Propagation delay single p-term</td>
<td>Min. 5.7 ns</td>
</tr>
<tr>
<td>(T_{PD2})</td>
<td>Propagation delay OR array</td>
<td>Min. 6.0 ns</td>
</tr>
</tbody>
</table>

Problem discussion
4.2 Designing an 8-bit adder/subtractor for integer numbers

Solve these basic addition and subtraction operations for integer numbers in two’s complement (2C) and 8 bits.

a) Indicate the result and the value of the overflow flag after performing the operations:
   - Addition: (-100) + (-15)
   - Subtraction: (+100) - (+6)
   - Subtraction: (+6) - (-127)
   - Addition: (-127) + (-100)

b) Draw the above operations, which are also examples of the circuit’s truth table, in a timing diagram. How long is the truth table?

Propose an internal architecture based on components and signals (plan C2) for the entity represented in Fig. 28. Follow the indications in P3 and P4 dedicated to arithmetic circuits.

c) How many VHDL files your circuit contains? Name them all.

d) Translate the top circuit schematics to VHDL and name it Int_add_subt_8bit.vhd. Find also the VHDL files for the components.

Development of the 8-bit integer adder/subtractor.

e) Start a project using an EDA tool and synthesise the circuit in a given target chip CPLD or FPGA.

f) Print the RTL view and the technology schematic and comment it.

Test the 8-bit adder/subtractor.

g) Start a VHDL simulator EDA tool and run a test bench to verify the unit under test (UUT) applying some 8-bit integers (positive and negative numbers) like the ones in a). Print the timing diagram and add notes and explanations to analyse the result.

h) Measure the maximum speed of operation (or computation or data processing) of the circuit using a gate-level simulation. Print the timing diagram explaining how such measurements are made. Explain the data from the time analyser spreadsheet.

i) Can you compare result when the 8-bit adder/subtractor is solved using 4-bit carry-lookahead instead of 4-bit ripple-carry adders?
4.3 Designing a 10-bit comparator for radix-2 and 2C numbers

The idea is to design and implement a 10-bit comparator for both, radix-2 (natural numbers including the zero, also named whole numbers) and two’s complemented (2C) numbers (integers, which are positive or negative). The entity symbol is represented in Fig. 29 and shows the data select input N used to set the input data type in operation: N = ‘1’ integers, N = ‘0’ radix-2. Let us name this project Selectable_comp_10bit.

The hierarchical multiple-VHDL file design strategy will follow the plan C2 studied in P3 based on using the component Comp_10bit that was inferred in the previous problem 3.9 for natural numbers in radix-2.

A good idea is to organise the project in several design phases:

A. Study the previous problem 4.3 on radix-2 comparators.
B. Study how to implement a comparison of two integers. Which may be the algorithm, or put in another way: how to use Comp_10bit to work with integers?
C. Study how to combine the previous designs into the final Selectable_comp_10bit chip.

1. Discuss the output of the circuit for several radix-2 numbers and for several 2C integer numbers. Draw the circuit’s truth table and a timing diagram sketch.
2. Discuss how to obtain a plan for this circuit based on the Comp_10bit component and other circuit.
3. Translate your schematic to VHDL and run the synthesis project for a given target CPLD or FPGA chip. View and comment the RTL and the technology schematics.
4. Verify the circuit applying several vectors using a VHDL testbench.
4.4 Performing gate-level simulations and propagation time measurements

This project takes as example circuits for performing gate-level or timed simulations the \textit{Adder\_1bit} and the \textit{Int\_add\_subt\_8bit} already developed earlier. Here the idea is to discuss which circuit is faster and how a circuit can have an optimised topology in order to be faster.

a) Perform a gate level simulation in an \textit{Adder\_1bit} structural circuit like the one represented in Fig. 30. More architectures can be found in the 1-bit full adder tutorial. Find the maximum speed of computation of this circuit for a given target PLD chip.

![Structural Adder\_1bit consisting of several levels of gates.](image)

b) Perform a gate level simulation in an \textit{Int\_add\_subt\_8bit} build using ripple carry techniques. Find the maximum speed of computation of this circuit for a given target PLD chip.

c) Perform a gate level simulation in an \textit{Int\_add\_subt\_8bit} build using carry-lookahead techniques. Find the maximum speed of computation of this circuit for a given target PLD chip.

d) Compare results for the same \textit{Int\_add\_subt\_8bit} when the target chip is a CPLD or a FPGA.
4.5 How to design an 8-bit multiplier for 2C integer numbers?

This is a complex project which go further beyond the objectives of CSD course. However, it has its interest in showing complex circuits can be organised systematically using our tools and VHDL plan C2.

a) Run simulations of the circuit in Fig. 31 using the multiplier available in Proteus. Determine the range of the numbers and the operation that can be handled using 8-bit 2C integers. Draw the circuit’s truth table for several example calculations.

b) Study how the architecture of a 8-bit hardware multiplier for natural numbers in radix-2 (Mult_8bit) is organised when using 1-bit multiplier cells (Mult_1bit).

e) Discuss the plan for building the 2C integer multiplier.

f) Study if there any alternative to our Plan C2 for drawing this huge architecture in VHDL (generics, scalable circuits, plan B, numeric library, etc.)

g) Browse the web to find example VHDL code which can be copied and adapted to our circuits. 1) Mult_8bit, 2) Selectable_mult_8bit.
Sequential systems
P5 1-bit memory cells: latches and flip-flops

Objectives

After studying the content of these problems, you will be able to:

- Implement the basic RS latch using NOR or NAND gates.
- Deduce a data flip-flop (D-FF) from an RS latch.
- Explain the concept of a clear direct (CD) and set direct (SD).
- Explain the concept of CLK signal.
- Describe the specifications of flip-flops (RS_FF, JK_FF, D_FF and T_FF): function table, state diagram, timing diagram and symbol.
- Analyse simple asynchronous circuits based on latches or flip-flops (for instance, the 7493 chip).
- Explain the idea of sampling input values (level-sensitive signals) and synchronicity.
- Explain the concepts of time resolution and glitch in a synchronous digital system.
- Analyse simple synchronous circuits based on flip-flops or latches.
- *Debounce* and synchronise digital signals from pushbuttons and switches.
- Find characteristics of classic (LS, HCT, etc.) 1-bit memory cell chips.
- Define the CLK to output delay time ($t_{CO}$).
- Develop projects in VHDL based on RS latches and flip-flops.
- Explain the VHDL description of a D_FF.
- Run functional and gate-level simulations to test and verify the performance of circuits based on flip-flops and latches.
- Connect a bank of latches or flip-flops to build $n$-bit memory cells.
- Implement logic functions using the method of ROM memories.
- Explain the initial idea of a finite state machine applied on flip-flops.

Project classification:

- RS latch and flip-flop conception: 5.1, 5.2
- Synchronous circuits: 5.3, 5.5
- Asynchronous circuits: 5.4, 5.8, 5.9
- FSM introduction: 5.6, 5.7
- ROM method for designing logic functions: 5.10, 5.11
5.1 Designing and using an RS latch. Deducing an RS_FF.

Invent an RS latch using NOR or NAND gates. This is a structural plan A based on logic equations. Run this tutorial to develop and simulate the circuit. Use the RS latch to debounce a SPDT switch. Follow this tutorial on how to implement an RS_FF from an RS latch using logic gates (plan A). However, the topic is theoretical or conceptual, to comprehend the functionality of the special CLK and CD signals, because the practical implementation of flip-flops in VHDL will be by means of behavioural plan B. These are tutorials and projects on D_FF, T_FF and JK_FF respectively.

![Symbol and function table of an RS latch.](image)

<table>
<thead>
<tr>
<th>R</th>
<th>S</th>
<th>Q^+ in the future</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>Q (the current value)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>L set to '1'</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>Q reset to '0'</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>Not permitted (undefined)</td>
</tr>
</tbody>
</table>
5.2 Data flip-flop (D_FF)

Follow this tutorial on how to implement a D_FF in VHDL.

1. Specifications (symbol, function table and example timing diagram). Find a commercial chip of this kind.

2. Plan B: state diagram. Flow chart to describe the state diagram, function table or algorithm. Write the D_FF.vhd from the flow chart file.

3. Run a project using an EDA synthesis tool for a CPLD or FPGA target chip. Print and discuss the RTL and the technology schematics.

4. Simulate the circuit using a VHDL test bench and discuss the results.

5. Use a gate-level simulation to measure the maximum speed of operation. Run the timing analyser and compare results.

Fig. 33
Symbol and function table of a data type flip-flop (D_FF).
5.3 Analysis of a synchronous circuit

Analyse the circuit in Fig. 34, by drawing a timing diagram of the outputs \( Q(3..0) \). Indicate a possible application of this circuit. How many VHDL files will be required to develop the project?

Fig. 34
Circuit based on data flip-flops (D_FF).

After the rising edge of the CLK

<table>
<thead>
<tr>
<th>D</th>
<th>( Q^+ )</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Whenever CD = 1  
\( \Rightarrow Q = 0 \) immediately

Some insight into the solution of the problem can be found here.
5.4 JK_FF and analysis of an asynchronous circuit

Component analysis

a) Analyse the behaviour of the JK flip-flop in Fig. 35 and represent the output Q in a timing diagram like the one represented in Fig. 36.

Fig. 35
Symbol and function table of a synchronous JK flip-flop.

<table>
<thead>
<tr>
<th>JK</th>
<th>Q^+</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>Q</td>
</tr>
<tr>
<td>01</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>Q'</td>
</tr>
</tbody>
</table>

Fig. 36
Example input waveforms.

CLK
CD
J
K
Q

Follow the tutorial P5 and study some theory on flip-flops and how they work. What could its internal structure be (derived from gates, using components like RS_FF and extra logic or using a standard FSM architecture based on a D_FF state register)? Why do we prefer an FSM architecture?

b) How can a toggle T flip-flop (T_FF) be designed using a JK flip-flop component? Explain the circuit.

c) Explain the internal architecture of the JK_FF circuit if it is designed as a finite state machine (FSM) that has the state diagram shown in Fig. 35. Design the logic of the CC1 and CC2 circuits using behavioural plan B as in the tutorial T_FF.

d) Explain the circuit required to implement the data type flip-flop (D_FF) in the state register from the initial RS_latch cell based on only-NOR gates. (See these class notes).
Asynchronous circuit analysis

e) Deduce the outputs of the circuit represented in Fig. 37. This is an example of an asynchronous circuit that can serve to demonstrate how complicated and unreliable the asynchronous design is compared to the synchronous canonical design based on FSM, which is presented in the next P6.

NOTE: to deduce the vector output \( Q(2..0) \), you must first draw the timing diagram waveforms using this procedure.

Circuit simulation in Proteus
f) You can verify your answer by comparing it to the Proteus circuit that can be obtained by modifying a similar circuit such as this one in Problem 5.8. Circuit development in VHDL and testing

Development and testing means that now the circuit in Fig. 37 is a conventional project, called for instance async.vhd, which must be planned carefully in VHDL from bottom to top. For instance:

1. Solve and test completely the JK_FF specified in P5 so that you get a JK_FF.vhd file that can be used in this project as a component.
2. Now that you know how the circuit works, you can write in VHDL the asynchronous circuit in Fig. 37, synthesise it and print the RTL view. Be aware that the “number of registers” in the project’s summary spreadsheet must be “3”.
3. Use a VHDL test bench to demonstrate that the timing diagram looks like that obtained in Proteus or in the analysis above.

Canonical circuit, a P6 project

g) Design an FSM that generates the same output Q(2..0). It will be a better replacement of the asynchronous circuit in Fig. 37. Why?
5.5 Analysis of a synchronous circuit

1. Analyse the circuit in Fig. 39 and, draw a timing diagram of the outputs Q(3..0) that apply to this procedure. How many states does this circuit have? Which is the output value Q(3..0) for each state?

2. Verify your results by simulating the circuit in Proteus. The schematic in Fig. 39 can be captured by modifying a similar circuit such as this one in Problem 5.8.
5.6 **Design a toggle flip-flop (T_FF) using the FSM strategy**

Design a T_FF using the FSM strategy. Follow this tutorial.

1. **Specifications (symbol, function table and example timing diagram).** Find a commercial chip of this kind, for instance a JK_FF with T = J = K.

2. **Plan B: FSM strategy.** Draw the state diagram. Draw the state register, the truth table of CC1 and CC2. Translate the truth tables of CC1 and CC2 into a flow chart (behavioural description). Write the T_FF.vhd.

3. **Run a project using an EDA synthesis tool for a CPLD or FPGA target chip.** Print and discuss the RTL and the technology schematics.

4. **Simulate the circuit using a VHDL test bench and discuss the results.**

5. **Measure the maximum speed of operation using a gate-level simulation.** Run the timer analyser tool and compare results.

---

**Fig. 40** Symbol and function table of a toggle type flip-flop (T_FF).
5.7 Design a JK flip-flop using the FSM strategy

Design a JK_FF using the FSM strategy. Follow this project organisation.

1. Specifications (symbol, function table and example timing diagram). Find a commercial chip of this kind.

2. Plan B: FSM strategy. Draw the state diagram. Draw the state register, the truth table of CC1 and CC2. Translate the truth tables of CC1 and CC2 into a flow chart (behavioural description). Write the JK_FF.vhd.

3. Run a project using an EDA synthesis tool for a CPLD or FPGA target chip. Print and discuss the RTL and the technology schematics.

4. Simulate the circuit using a VHDL test bench and discuss the results.

5. Measure the maximum speed of operation using a gate-level simulation. Run the timer analyser tool and compare results.
5.8 Analysis of an asynchronous counter (type 7493)

Analyse the circuit in Fig. 43 and represent the waveforms in a diagram like the one represented in Fig. 44. Be aware that the circuit is asynchronous because chips’ CLK inputs are not connected to the same signal. Remember that a T-type flip-flop behaves as indicated in Fig. 42.

What is the circuit’s function? What will the circuit be used for? What is the circuit’s main problem, so that it must be rejected for precision applications?

+ A VHDL design tutorial of a similar circuit can be found here.
5.9 Analysis of an asynchronous circuit based on T_FF

Analyse the output waveforms and deduce the binary codes K(3..0) that generate the asynchronous circuit in Fig. 45 based on toggle flip-flops (T_FF).

Fig. 45
Diagram and function table of a T_FF and example of an asynchronous circuit.

Fig. 46
Output waveforms to be deduced from the circuit in Fig. 45.

A tutorial solution of the problem can be consulted here.

Optionally, the project can be continued developing and testing the circuit in VHDL.
5.10 Design a combinational circuit using the method of ROM memories

Tutorial at this link.
5.11 Design a HEX_7seg using the method of ROM memories

Tutorial at this link.
P6 Finite State Machines (FSM)

Objectives
After studying the content of these projects, you will be able to:

- Discuss about the standard architecture of a finite state machine (FSM): current and next states, the state register, and the output logic (CC2) and state logic (CC1) combinational circuits.
- Describe the truth table of the output logic circuit and find its behavioural interpretation (flow chart or algorithmic state machine (ASM) chart).
- Describe the truth table of the state logic circuit and find its behavioural interpretation (flow chart).
- Explain the specifications of the system: symbol, inputs and outputs, number of states and state transitions, state encoding (binary sequential, Gray, one-hot, etc.), state register (number of D_FF used in the design).
- Develop the FSM in a single (flat) VHDL file.
- Run functional and gate-level simulations to test and verify the FSM performance and characterisation.
- Design simple FSM using the CSD systematic methodology: from the specifications to the final verification and prototyping. For instance: traffic light sequencers, light control systems, matrix keypad encoders, step motor controller, push-button debouncer and synchroniser, etc.
6.1 Controlling the classroom luminaires

This is the light control (Light_Control) tutorial to introduce the FSM basic concepts.

+ A complete tutorial can be found here.
6.2 Invent a bicycle torch
This is the lamp control (Lamp_Control) tutorial to introduce the FSM basic concepts.

+ A complete tutorial can be found here.

6.3 Debouncing circuit
To get rid of signal electrical noise when using switches and push-buttons.

+ A complete tutorial can be found here.
6.4 16-key matrix encoder

To save cables and simplifying the interface for large keyboards.

A complete tutorial can be found here.
6.5 Water tank controller

a. Study the specifications. We want to design a water tank controller (Water_tank_controller) as an FSM that can drive two pumps independently, as represented in Fig. 47. The tank has level sensors D1, D2, and D3 attached to the wall, so that a ‘1’ is generated when the sensor is sunk into water. The controller works as follows: when it is empty, below D1, both pumps work simultaneously; when the water level is above D2 pump P1 stops; when the water is above D3, meaning that the tank is full, pump P2 stops; and finally, the pumps do not switch on until the water level is again below D1.

b. Plan: FSM. Draw the state diagram if, in addition to controlling the water level, we also want to indicate in a LED column the current level of the water in the tank.

c. Plan: adapt the general FSM architecture to this problem and draw the state register based on D_FF. Deduce how many D_FF are required if you are coding in binary sequential or in one-hot.

d. Plan: write the truth table of CC1 and CC2 and their equivalent behavioural interpretations in flowcharts.

e. Development: write the VHDL file Water_tank_controller.vhd by translating the flowcharts and the state register. Run a project using an EDA synthesis tool for a CPLD or FPGA target chip. Print and discuss the RTL and the technology schematics. The CLK oscillator is 1 MHz.

f. Test: simulate the circuit using a VHDL test bench and discuss the results. Measure the maximum CLK frequency that can be applied to your design considering a target chip from Lattice Semiconductor (ispMach4128V TQFP100), Intel (Cyclone IV EP4CE115F29C7), or Xilinx (Spartan-3E XC3S500E-FG320).
g. Additional features added to the basic prototype. The user wanted to add an extra circuit to translate the LED column code into a 7-segment display. Thus, an additional combinational circuit CC3 is required to meet this new specification. Let us solve the problem using the ROM method for implementing logic functions. The wiring in Fig. 48 shows the naming conventions for the vector HEX0(6..0) common anode in the DE2-115 board user guide page 36. Discuss the size \([2^m \times n]\) of the ROM, its content and synthesise the circuit.
6.6 Traffic light controller

This problem is discussed in the tutorial.
6.7 Stepper motor controller

From an exam or problem in the digsys.
6.8 7-segment digit sequencer

We want to design a simple driver to shown a sequence of movement, clockwise and counter-clockwise, in a single 7-segment display. Fig. 49 represents the schematic diagram of the application. The circuit components are: (1) a clock to produce a rectangular wave with a given frequency, let us take for instance 5 Hz; (2) the digital system named sequencer and (3) the 7-segment display (common cathode) with its current-limiting resistors.

![Diagram](image)

The system has to work as specified in Fig. 49c, depending on the logic levels of the synchronous input signals: UD_L (Up –active high / Down –active low) and ST (start/stop). A start pulse (ST) activates the sequence of LED lighting that never ends until another pulse ST is applied and the sequence reached the last state. Because of the requirement that the sequence must end (for example when going UP reaching the state Blank) before stopping (going Idle) if another ST pulse is detected, the design must include a 1-bit memory cell such as an RS_Latch or an RS_FF and the FSM that generate the sequence and controls the system. Therefore, this will be a plan C2 system composed of a top design (sequencer) and some components.

1. Deduce a circuit for solving this problem. This is an initial discussion.
2. Particularise the internal FSM component architecture to this problem, naming and connecting all the inputs and outputs. How many D_FF of memory are used in this problem if coding the state machine in one-hot?
3. Infer and draw the circuit’s state diagram. Annotate all the state transitions and outputs.
4. Sketch a timing diagram showing the main operations. In addition to the ports, include as well internal signals like STB in the discussion.
5. Draw the state register if coding the machine in binary sequential.
6. Write the CC2 truth table to obtain the outputs of the circuit and its flow chart.
7. Design the CC1 truth table to obtain the next state to go and its flow chart.

------------------------ Development and test ------------------------
8. Write the VHDL files (this is a plan C2 design) and run the EDA project to synthesise the circuit and obtain results. Inspect the RTL and verify that it looks like your schematic. Check the number of D_FF, print and comment the schematics.
9. Write a VHDL test bench and run the EDA simulation tool to verify your design.
10. The target chip is the ispMACH4128 which has DFF with a $t_{co} = 2.7$ ns and logic gates with a $t_{pd} = 2.7$ ns. Which may be a good estimation of the maximum frequency of operation? Explain your answer.

Extra (P8 content on CLK generators: counters and frequency dividers using the plan Y)
11. Design a circuit to produce the 5 Hz square wave from a 50 MHz quartz crystal oscillator and deduce the number of D_FF that will contain.
P7 Standard counters and registers

Objectives

After studying the content of these projects, you will be able to:

- Draw synchronous timing diagrams for counters and registers.
- Explain the meaning and use of signals such as CE (count enable), LD (parallel load), TC (terminal count), Q (coded outputs), D_in (parallel data input), UD_L (up and down counting or reversibility), etc.
- Explain counters’ entity definitions and their functions tables.
- Design standard counters as another application of FSM.
- Design any kind of counter selecting one or several of the following VHDL strategies:
  - Plan X: as a typical enumerated FSM in P6 (single file).
  - Plan Y: as a scalable block based on STD_LOGIC_VECTOR signals and arithmetic VHDL library (single file).
  - Plan C2: using building blocks like standard counters and other components as in Chapter 1 (multiple file).
- Design data registers that are scalable to n bits.
- Design shift registers that are expandable to n bits.
- Design applications of sequential systems using registers and counters as building blocks.
- Read and interpret classic counters datasheets.
7.1 1-digit BCD counter

This is the design of the 1-digit BCD counter represented in Fig. 50 using a flat design (1 VHDL file) based on plan X (state enumeration).

This example is the introductory project to discover the world of digital counters and their applications. Plan X is merely a continuation of P6 on FSM.

See the P7 project tutorial.
7.2 Synchronous universal 4-bit binary counter

Our goal is to design a very versatile building block that can be used in other designs as a component. It is a synchronous, presettable (LD, parallel load), 4-bit (modulo 16), reversible (up and down), binary counter with count enable (CE) and terminal count (TC16), as represented in Fig. 51. It is a chip similar to the classic 74LS169.

Fig. 51 Synchronous 4-bit universal binary counter. Symbol and function table. This is an example of a Proteus simulation of a very similar circuit.

<table>
<thead>
<tr>
<th>LD</th>
<th>CE</th>
<th>UD_L</th>
<th>Q*</th>
<th>Synchronous operation after the CLK’s rising edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>Din</td>
<td>Parallel load (register data)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>x</td>
<td>Q</td>
<td>Do nothing (inhibit)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(Q+1)_{mod16}</td>
<td>Up counting in binary</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(Q-1)_{mod16}</td>
<td>Down counting in binary</td>
</tr>
</tbody>
</table>

TC16 = ‘1’ when CE = ‘1’ and [(Q = 15 and UD_L = ‘1’) or (Q = 0 and UD_L = ‘0’)]; ‘0’ otherwise

Specifications

1. Symbol, function table, example of a timing diagram. Example of a commercial circuit. Meaning and descriptions of inputs and outputs.

Planning (plan Y)

2. Draw a sketch of the state diagram. How many states must this sequential system have?
3. Customise the general FSM architecture for this problem, indicating where every input and output is connected. Plan the circuit as an FSM in a single VHDL file.
4. How many data flip-flops (D_FF) are required? Draw the schematic of the state register. What is the internal encoding of the current_state signal?
5. Name the circuit Counter_mod16 and plan the circuit as an FSM in a single VHDL file.
6. Write the truth table for the CC2 and generate a flowchart as a behavioural interpretation.
7. Write the truth table for the CC1 and generate a flowchart as a behavioural interpretation. Propose an internal circuit for instance using MUX.

Developing

8. Translate the circuit schematic to VHDL and start an EDA project to synthesise it for a given CPLD/FPGA target chip.
9. Print the RTL view and technology schematics. Inspect and analyse the circuits annotating comments using pens on the schematics. Check the number of D_FF registers.
Testing and verifications

10. Draw the sketch of the testbench fixture. Run a functional simulation once the timing diagram in the specifications is translated into a VHDL test bench. Print the waveforms from the logic analyser and annotate using pens on the waveforms.

11. Run a gate-level simulation and determine propagation time from CLK to output ($t_{CO}$) at a given transition. Print the waveforms from the logic analyser and annotate using pens on the waveforms.

12. Run the timing analyser tool, and thus calculate the maximum frequency of operation for the given target chip.

Problem discussion. This picture is an example of a functional simulation of the universal counter modulo 16. The comments in red ink are very important, to check whether the circuit works as expected. Use coloured pens and handwriting to discuss the results instead of text boxes.
7.3 Synchronous modulo 12 counter

Follow the usual steps to design as a complete project the synchronous up/down modulo 12 counter represented in Fig. 52 using two different strategies. Compare and discuss the advantages and drawbacks of each strategy.

Project X. Single VHDL file (flat) FSM based on naming states and the use of State_type enumerated signals. Using this methodology, the project becomes simply another P6 FSM example.

Project Y. Single VHDL file (flat) FSM based on the arithmetic VHDL library and the use of std_logic_vector signals. Using this methodology, the project becomes another exercise like the Counter_mod16 in Problem 7.2.

Project C2. Hierarchical structure (multiple file project) based on the building block Counter_mod16 engineered in Problem 7.2.

Problem discussion and project files.
7.4 Data register

Design a synchronous 24-bit data register shown in Fig. 53 using plan Y. How many states have this circuit?
Follow the usual steps stated in Problem 7.2.

Here is a tutorial on the design of a 4-bit data register.
7.5 Shift register

Design a synchronous 8-bit universal shift register using plan Y. How many states have this circuit? How can three `shift_reg_8bit` blocks be connected to build a 24-bit shift register (plan C2)?

Fig. 54 Synchronous 8-bit shift register.

Here is a tutorial.
## 7.6 Hour counter for a real-time clock

Our goal is to design an hour counter for use in a real-time clock device to count the hours in modes 0–12 (M = ‘1’) and 0–24 (M = ‘0’). Fig. 55 represents the schematic diagram of the application when it is connected to 7-segment digits.

![Schematic diagram](image)

**Specifications**
1. Explain the function table of the `hour_counter` and discuss the modes of operation.
2. Draw an example of a timing diagram. How many states will the hour counter contain?
3. Draw the function table and symbol of a synchronous 4-bit binary universal counter (`Counter_mod16`).

**Plan**
4. Organise the internal architecture of the hour counter based on the use of universal 4-bit binary counters (`Counter_mod16`) and combinational circuits and logic gates.
5. How many VHDL files will be required to complete the `Hour_counter_top` in Fig. 55?

**Develop**
6. Find the `Counter_mod16.vhd` file and translate the hierarchical schematic in Fig. 55 to VHDL.
7. Start a synthesis project and inspect the RTL and technology view schematics. Check the project summary to verify the number of DFF. The target chip may be any CPLD or FPGA that is available in the laboratory.

**Testing**
8. Translate the timing diagram in 2) to VHDL (`Counter_mod16_tb.vhd`) and run the functional test.
9. Run a gate level simulation and measure the $t_{CO}$ parameter and thus the maximum speed of operation.

Prototype
10. Choose a laboratory experimentation board like NEXYS 2 from Digilent. Assign pins using the constraint editor, and build and check the prototype `Hour_counter_top`, adding the necessary chips and modifications. Pay attention to how the 7-segment digits are wired.

Problem discussion. This picture is an example of a functional simulation of the `Hour_counter` working in AM-PM mode ($M = '1'$).
## 7.7 6-bit binary universal counter

### Specifications

Our aim is to design the block represented in Fig. 56, a 6-bit synchronous binary counter *(Counter\_mod64)* fully equipped with many features to make it versatile as a component in many projects. Another example on how the architecture developed in Problem 7.2 following plan Y can be easily expanded.

Fig. 56. The sequential block to be designed and its function table explaining all the synchronous modes of operation.

#### Function Table

<table>
<thead>
<tr>
<th>LD</th>
<th>RST</th>
<th>CE</th>
<th>UD_L</th>
<th>Q(^+)</th>
<th>Synchronous operation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>Din</td>
<td>Parallel load (register data)</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>0</td>
<td>Reset</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>x</td>
<td>Q</td>
<td>Do nothing (inhibit)</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>(Q+1_mod64)</td>
<td>Up counting in binary</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>(Q-1_mod64)</td>
<td>Down counting in binary</td>
</tr>
</tbody>
</table>

TC64 = ‘1’ when CE = ‘1’ and [(Q = 63 and UD\_L = ‘1’) or (Q = ‘0’ and UD\_L = ‘0’)]; ‘0’ otherwise.

1. Draw state diagrams for the circuit to describe the counter’s modes of operation. How many states will this FSM contain?
2. What is the difference between the synchronous RST signal and the CD?
3. Draw a timing diagram to represent the modes of operation: Load, RST, count UP, count DOWN and do nothing.
Planning as an FSM in a single VHDL file (plan Y)
4. Draw the architecture of the FSM adapted to this problem and indicate where all the inputs and outputs are connected.
5. How many registers (D-type flip-flops) will the system include? Explain your answer.
6. Draw the truth table and flow chart of the combinational circuit CC1.
7. Draw the truth table and flow chart of the combinational circuit CC2.

Development using CAD/EDA tools
8. What is the synthesis process? In what way can we examine and discuss the results of the synthesis process?

Testing and verification using EDA VHDL simulation tools
9. Draw the sketch of the VHDL testbench fixture of interconnections. Write the main features of a test bench file Counter_mod64_tb.vhd that is required to perform a functional simulation. For instance, translating to VHDL some vectors from the 3rd section.
10. What are the “VHO” and the “SDF” files? What are these files used for? The target chip is ispMACH4128V, which has D_FF with a $t_{CO} = 2.7$ ns and logic gates with a $t_{PD} = 2.7$ ns. What could be a good estimation of the maximum frequency if operation? Explain your answer. Naturally, the section can be solved for other target PLD chips.
7.8 Designing a Johnson counter

Design a synchronous 5-bit Johnson counter with count enable (CE) and reversibility (UD_L) control signals, as shown in the symbol in Fig. 57 using VHDL approaches. For instance:
- Plan X: single-file, FSM strategy enumerating and labelling states
- Plan C2: multiple-file, using a standard component like the `Counter_mod16` developed in Problem 7.2.

![Symbol of the synchronous sequential system to be invented in this project.](image)

Specifications

1. Draw the Johnson code for 5 bits. Draw the function table for this counter. Draw an example of a timing diagram. Find and study an industry standard circuit based on classic technologies such as the 74HCT4017. In Chapter 3, the same circuit is solved in Problem 10.4 using a microcontroller.
2. Draw the state diagram indicating both and outputs.

Plan X:

3. Draw the FSM structure consisting of CC1 and CC2 and the state register. Indicate its inputs and outputs. How many D_FF will the state register contain if the internal states are coded in binary and if they are coded in one-hot?
4. Draw the CC1 truth table and its equivalent behavioural representation in a flowchart.
5. Draw the CC2 truth table and its equivalent behavioural representation in a flowchart.

Development. Target chip Intel Cyclone IV EP4CE115F29C7N

6. Write the VHDL code by copying and adapting a similar example from the digsys web. Run the EDA synthesis to inspect the RTL and technology views. Print them both and annotate the comments using pens and handwriting. Check the number of D_FF used.
Testing and verification

7. Draw the sketch of the VHDL testbench fixture. Use a VHDL stimulus testbench file from the timing diagram in section 1 and run an EDA functional simulation to check how the circuit behaves over time. Print the waveforms and annotate them using pens and handwriting.

What is the maximum frequency that can be assigned to the CLK signal when a functional simulation is performed?

8. Run a gate-level simulation and measure the propagation time from CLK to output \((t_{CO})\) in a given transition. Print the waveforms and annotate those using pens and handwriting.

9. Run a timing analyser and calculate the maximum frequency that can be assigned to the CLK signal. Calculate the same parameter if the target chip is an Intel CPLD Max II EPM2210F324C3 that has the following characteristics:

<table>
<thead>
<tr>
<th>MAX II Device Features</th>
<th>EPM2210</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_{PP} (\text{ns}))</td>
<td>1.7</td>
</tr>
<tr>
<td>(t_{CO} (\text{ns}))</td>
<td>4.6</td>
</tr>
</tbody>
</table>

Plan C2:

10. Propose an internal architecture based on \(\text{Counter}_{mod16}\) and other combinational blocks. Study the Problem 7.3. It is a good idea to design the counter with only an up counting direction first. Then, complete the counter by adding the reversibility feature. How many D_FF will this counter include?

Optional. Only if the project has to be developed:

11. Annotate completely the schematic, chips, signals, etc. and leave it ready for VHDL translation.

+ Problem discussion using Plan X.
P8 Dedicated processors and advanced circuits

Objectives

After studying the content of these projects, you will be able to:

- Explain the concept of a datapath or operational unit.
- Explain the need and the functionality of the control unit based on a FSM.
- Design complex circuits or dedicated processors based on datapath and control unit.
- Design CLK generators to obtain from a high frequency quartz crystal the many CLK signals required for the project under construction.
- Explain how to design frequency dividers.
- Explain how to square pulsed signals using T_FF.
- List a number of applications and complex circuit that can be designed using the techniques and strategies described in this P8 section.
- Explain why a microprocessor is a programmable dedicated processor.
8.1 Generation of CLK signals

The sketch in Fig. 58 represents the internal architecture of the \textit{CLK\_generator} block that was built to obtain the required CLK signals for a traffic light controller FSM when connected to the UP2 board quartz crystal oscillator of 25.175 MHz.

a) How many DFF registers will require the circuit?

b) How many VHDL files will require the implementation of the circuit?

c) How can you speed up the simulation of the circuit using an EDA tool?

d) Explain how to plan and design a similar circuit to obtain the squared signals of 2.5 kHz (CLK\_2\_5kHz\_SQ) and 10 Hz (CLK\_10Hz\_SQ) from an OSC\_CLK\_in of 24 MHz.

e) Explain how to design a circuit like the Chip 2 FREQ\_DIV\_25 in Fig. 58 using VHDL and the FSM technique (plan Y). Represent a timing diagram to show how it works.

f) Write the VHDL code for the circuit and implement the system using EDA tools.
8.2 Pulse generator

In Fig. 59 there is the symbol of a synchronous sequential machine to generate a burst of digital pulses. The proposed circuit is an adaptation to CSD of the original idea from this book [2]. The timing diagram in Fig. 60 represents how a number of pulses (0, 1, 2 or 3) are generated after triggering the Start_PB input. It can also be seen how an end of operation flag (EO_Flag) is issued to indicate that the machine is no longer occupied and can be triggered again.

The architecture in Fig. 61 fulfil the specifications; it is based on a FSM (Chip1) and other components. Why a component such a synchronous data register (Chip2) is necessary? Why a circuit like the debouncing filter (Chip4) is required to interface the Start push-button?

Explain the internal architecture of the Data_register_2bit component. How many states does it have? How many data flip-flops (DFF) are required to implement its state register?

Infer the state diagram of the Pulse_Gen_FSM that may solve the sequence of operations to generate the outputs.

Draw the CC2 truth table and its flowchart interpretation, so that it can be coded in VHDL in the usual CSD style.
Draw the CC1 truth table and its flowchart interpretation, so that it can be coded in VHDL in the usual CSD style.

Draw the internal circuit of the state register. How many data flip-flops (DFF) are required to implement it if we encode the machine in binary (sequential), or alternatively in one-hot?

Write down the VHDL test bench translating approximately the inputs signals in the Fig. 60 diagram.

If the FPGA used as a target chip to synthesise the system has a worst-case time to output propagation delay ($t_{CO}$) of 5.6 ns, which is the maximum CLK frequency and so the minimum pulse duration?

If the circuit uses a 16 MHz $OSC_{CLK\_in}$ from a quartz crystal, invent a CLK generator block to produce both square signals, a system CLK of 10 kHz and a 100 Hz signal to drive the debouncing circuit. How many VHDL files may it contain?
8.3 Designing an industrial application

In a gym and fitness centre, there are some shower stalls like the one represented in Fig. 64 that have to be automated to generate cycles of contrasting hot (48 °C), warm (26°C) and cold (4°C) water sprays simply clicking a single start push button (SB). After clicking the SB, initially warm water flows for 50 s (H = C = ‘1’), then hot water (H = ‘1’, C = ‘0’) for 10 s, and thirdly cold water (H = ‘0’, C = ‘1’) for 20 s, and this cycle is repeated another time; finally, the system goes idle (H = C = ‘0’) to wait for another user service. During the operation the R_LED turns on and the water solenoid valve (SV) is on. Let us design the digital control system connected to the valves’ power driver (Chip5) using two technologies: a CPLD/FPGA and a microcontroller.

Fig. 62 Photograph and sketch of the shower installation to be automated.
1. Explain the function of the circuit Chip1, why it is required for conditioning the start button external signal.

With respect to the Chip4:
2. Draw the state diagram of the application explaining both, state transitions and outputs in each state.
3. Draw an example of a timing diagram.
4. Draw the architecture of a FSM for the Chip4 explaining where all the inputs and outputs are connected.
5. Draw the CC2 truth table and its flowchart interpretation, so that it can be coded in VHDL in the usual CSD style.
6. Draw the CC1 truth table and its flowchart interpretation, so that it can be coded in VHDL in the usual CSD style.
7. Draw the internal circuit of the state register. How many bits and DFF (data-type flip-flops) will be used if the states are coded in one-hot?

With respect to the Chip3:
8. Explain the internal architecture, components and the number of VHDL files of the Prog_Timer project.

With respect to the Chip2:
9. Deduce and explain the internal architecture, the number of VHDL files and names of the CLK_Generator project.
10. We have measured by means of a gate-level VHDL simulation for the target CPLD/FPGA where the circuit is synthesised, a worst-case CLK to output propagation delay ($t_{CO}$) of 6.3 ns. Thus, which is the maximum $OSC_{CLK\text{\_in}}$ frequency and so the minimum pulse duration?
8.4 Design a 2-digit even/odd counter with start/stop button

The idea is to design a 2-digit BCD counter that counts even or odd numbers. Fig. 63 shows the main ideas of the specifications. Using the same ST button for start and stop operations makes it possible to plan the systems as an advanced circuit based on datapath and control unit to better handle counting operations.

These is a feasible plan that relies on previous CSD components and projects. You have two options to design the binary Counter_mod50:

a. Using the plan Y in a single VHDL file.
b. Using a plan C2 and components Counter_mod16 and other logic.

8.5 Synchronous serial adder

This is the project to review and write as a problem: serial adder (ref.)
8.6  **Timer MMSS**
This is the project proposed in P8.

8.7  **Synchronous serial multiplier**
This is the project to review and write as a problem: serial multiplier (ref.)

8.8  **Serial transmitter and receiver (USART)**
This is the project to review and write as a problem: USART (ref.)
8.9 Steeping motor control based on a dedicated processor

This is the project to review and write as a problem: (ref.)
Microcontroller applications

P9 Basic theory on microcontrollers (µC) and basic digital I/O interface

Objectives
After studying the content of this chapter on the basics of microcontrollers, you will be able to:

- Explain why there are 8-bit, 16-bit, 32-bit families of microcontrollers
- Explain the basic architecture of an 8-bit microcontroller: ALU, working register (accumulator), configuration registers, RAM memory, I/O ports, program memory, Harvard vs. Von Neumann architectures, etc.
- Design a combinational circuit using a microcontroller interfacing the digital I/O ports for inputs and outputs.
- Capture schematics containing microcontroller chips in Proteus (hardware planning).
- Comprehend the idea of a hardware/software diagram.
- Organise the software plan using the hardware interface functions: `init_system()`, `read_inputs()`, `write_outputs()`, and the data processing function `truth_table()`.
- Start a software project using the IDE tool and the C compiler.
- Translate the diagrams and flowchart to C language to write the C source files.
- Build software projects for a given target microcontroller in order to obtain executable files (.cof, .hex).
- Simulate applications based on microcontrollers using Proteus.
- Watch RAM variables and execute step by step instructions for debugging purposes.
9.1 The microcontroller PIC16F

Answer the following questions referred the microprocessors in general:

1. Explain the differences between Harvard (Microchip PIC) and Von Neumann (Intel 8051) microprocessor architectures. Draw the sketch of the architectures.
2. Which is the main architectural difference between 8/16/32 bits microprocessors?
3. Which are the functions of the FLASH (ROM) memory and the RAM registers?
4. Explain what the stack memory is and how it is used for.
5. Describe the main blocks of the central process unit (CPU) and how can you connect it to the content of the previous Chapters 1 and 2.
6. How an assembler instruction is executed? Find an example of C code disassembled and explain how it works.
7. How many clock cycles are required for executing an instruction in assembler?

The architecture of the PIC16F87xA family is presented in Fig. 64.
- Examine it and list the components that you could be able to design and synthesise, if that were the case, into a PLD using VHDL and applying strategies from previous chapters.
- Can you redraw the architecture as a programmable dedicated processor to solving each machine-code instruction?
- Draw the blocks of the RAM and the ROM components and explain how to perform memory writing and reading operations.
- Find the specification of the Timer0 peripheral and compare them with the programmable timer designed in Chapter 2.
Fig. 64 PIC16F87x architecture.
## Problems on digital circuits and systems

### BYTE-ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Cycles</th>
<th>14-Bit Opcode</th>
<th>Status Affected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDWF f, d</td>
<td>Add W and f</td>
<td>1</td>
<td>00 0111 dffe fffe</td>
<td>C,DC,Z</td>
<td>1,2</td>
</tr>
<tr>
<td>ANDWF f, d</td>
<td>AND W with f</td>
<td>1</td>
<td>00 0101 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>CLRWF</td>
<td>Clear f</td>
<td>1</td>
<td>00 00ef 0ffe fffe</td>
<td>Z</td>
<td>2</td>
</tr>
<tr>
<td>CLRWF -</td>
<td>Clear W</td>
<td>1</td>
<td>00 00ef 0000 fffe</td>
<td>Z</td>
<td>2</td>
</tr>
<tr>
<td>COMWF f, d</td>
<td>Complement f</td>
<td>1</td>
<td>00 1001 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>DECFSZ f, d</td>
<td>Decrement f, Skip if 0</td>
<td>1</td>
<td>00 1011 dffe fffe</td>
<td>Z,1,2,3</td>
<td></td>
</tr>
<tr>
<td>INCFSZ f, d</td>
<td>Increment f, Skip if 0</td>
<td>1</td>
<td>00 1010 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>MOVWF f, d</td>
<td>Move f</td>
<td>1</td>
<td>00 1000 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>MOVWF f</td>
<td>Move W to f</td>
<td>1</td>
<td>00 0000 lffe fffe</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NOP</td>
<td>No Operation</td>
<td>1</td>
<td>00 0000 0000 0000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RLF f, d</td>
<td>Rotate Left f through Carry</td>
<td>1</td>
<td>00 1101 dffe fffe</td>
<td>C</td>
<td>1,2</td>
</tr>
<tr>
<td>RRF f, d</td>
<td>Rotate Right f through Carry</td>
<td>1</td>
<td>00 1100 dffe fffe</td>
<td>C</td>
<td>1,2</td>
</tr>
<tr>
<td>SUBWF f, d</td>
<td>Subtract W from f</td>
<td>1</td>
<td>00 0010 dffe fffe</td>
<td>C,DC,Z</td>
<td>1,2</td>
</tr>
<tr>
<td>SWAPF f, d</td>
<td>Swap nibbles in f</td>
<td>1</td>
<td>00 1110 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
</tr>
<tr>
<td>XORWF f, d</td>
<td>Exclusive O R W with f</td>
<td>1</td>
<td>00 0110 dffe fffe</td>
<td>Z</td>
<td>1,2</td>
</tr>
</tbody>
</table>

### BIT-ORIENTED FILE REGISTER OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Cycles</th>
<th>14-Bit Opcode</th>
<th>Status Affected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>BCF f, b</td>
<td>Bit Clear f</td>
<td>1</td>
<td>01 00bb bffe fffe</td>
<td></td>
<td>1,2</td>
</tr>
<tr>
<td>BSF f, b</td>
<td>Bit Set f</td>
<td>1</td>
<td>01 01bb bffe fffe</td>
<td></td>
<td>1,2</td>
</tr>
<tr>
<td>BTFSC f, b</td>
<td>Bit Test f, Skip if Clear</td>
<td>1 (2)</td>
<td>01 10bb bffe fffe</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>BTFSS f, b</td>
<td>Bit Test f, Skip if Set</td>
<td>1 (2)</td>
<td>01 11bb bffe fffe</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

### LITERAL AND CONTROL OPERATIONS

<table>
<thead>
<tr>
<th>Mnemonic, Operands</th>
<th>Description</th>
<th>Cycles</th>
<th>14-Bit Opcode</th>
<th>Status Affected</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDLW k</td>
<td>Add Literal and W</td>
<td>1</td>
<td>11 11lx kkkk kkkk</td>
<td>C,DC,Z</td>
<td></td>
</tr>
<tr>
<td>ANDLW k</td>
<td>AND Literal with W</td>
<td>1</td>
<td>11 1001 kkkk kkkk</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>CALL k</td>
<td>Call Subroutine</td>
<td>2</td>
<td>10 0kkk kkkk kkkk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CLRWDT -</td>
<td>Clear Watchdog Timer</td>
<td>1</td>
<td>00 0000 0110 0100</td>
<td>TO,PD</td>
<td></td>
</tr>
<tr>
<td>GOTO k</td>
<td>Go to Address</td>
<td>2</td>
<td>10 1kkk kkkk kkkk</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>INCLW k</td>
<td>Inclusive OR Literal with W</td>
<td>1</td>
<td>11 1000 kkkk kkkk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MOVLW k</td>
<td>Move Literal to W</td>
<td>1</td>
<td>11 00xx kkkk kkkk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RETIFIE -</td>
<td>Return from Interrupt</td>
<td>2</td>
<td>00 0000 0000 1001</td>
<td>TO,PD</td>
<td></td>
</tr>
<tr>
<td>RETLW k</td>
<td>Return with Literal in W</td>
<td>2</td>
<td>11 01xx kkkk kkkk</td>
<td>Z</td>
<td></td>
</tr>
<tr>
<td>RETURN -</td>
<td>Return from Subroutine</td>
<td>2</td>
<td>00 0000 0000 1000</td>
<td>TO,PD</td>
<td></td>
</tr>
<tr>
<td>SLEEP -</td>
<td>Go into Standby mode</td>
<td>1</td>
<td>00 0000 0101 0011</td>
<td>TO,PD</td>
<td></td>
</tr>
<tr>
<td>SUBLW k</td>
<td>Subtract W from Literal</td>
<td>1</td>
<td>11 11lx kkkk kkkk</td>
<td>C,DC,Z</td>
<td></td>
</tr>
<tr>
<td>XORLW k</td>
<td>Exclusive O Literal with W</td>
<td>1</td>
<td>11 1010 kkkk kkkk</td>
<td>Z</td>
<td></td>
</tr>
</tbody>
</table>
9.2 Invent a Dual_MUX4 based on a $\mu$C

This assignment and tutorial can be found here.
9.3 1-digit BCD adder

The specifications of this project are simply add two 1-digit BCD numbers considering as well the $C_{in}$ and the $C_{out}$ signals to be able to chain components of the same kind. The circuit to solve is represented in Fig. 66.

Remember that, as usual, you have to organise the documentation to hand in in 4 sections, each one in different sheets of paper. Section 5, prototyping, is always optional in case you like to invest some more time in the laboratory downloading the microcontroller configuration program (hex) to the training board while measuring and characterising the prototype using workbench instrumentation.

The problem is reviewed and assessed in this way: half of the project, sections 1 and 2, is prepared in classrooms using paper and group discussions. The last sections 3 and 4 are solved by means of the virtual laboratory (IDE – Simulator) available from our virtual desktop computers. Remember that each of you have access to your personal network disk (L:) to properly develop and test your own project.

- Specifications and planning $\rightarrow$ 5p.
- Development and test $\rightarrow$ 5p.

A note on group discussions. Learning to work cooperatively is not an easy task, but indeed, very demanding. It doesn’t mean that one of you has to do all the work and the others simply have to copy and paste. On the contrary, use cooperation and group dynamics to clarify your ideas and to organise the project. Thus hand in an individual report (you are the responsible and the only author of all the material):

- Handmade specifications, tables, symbols, diagrams, timing diagrams, bullet list, etc.
- Handmade schematics, block diagrams, annotations, flowcharts, comments, etc.
- Write your own code and draw your own schematics. Print and comment schematics and C code. This is how you can print correctly your C files.
- Print the project compilation results to shown how your IDE has generated no errors. Annotate how many RAM memory bytes has been used and discuss whether there is an agreement with your initial planning of variables. Explain how long is your Flash program. Explain the difference between the COF and the HEX configuration files. Print a portion of disassembled code and explain details on how the data is transferred or the operations are executed. Etc.
- Print test results with meaningful annotations and discussion on the results to demonstrate that the project works as expected. Measure how long does it take to run the main loop code. Measure the time required to execute a single assembly instruction.

And on top of that, be fair: you are here at this university to learn the content of this subject deeply, not superficially. A hardware engineer has to be able to project circuits professionally and successfully, and to meet this goal, it is required a complete personal involvement and engagement. You can count as well with our commitment to help you any time.

A tutorial on the project can be read here.
9.4 12-to-4 encoder

The project objective is to design an encoder $Enc_{12\_4}$ like similar to the one presented in problem 2.4 using a µC. For instance, the application can be integrated as a subsystem in a professional PBX door phone with dialling keypad as represented in Fig. 67. The device has to generate the 4-bit code of the clicked key. BCD codes for keys from 0 to 9, and “1010” for the hash symbol “#” and the “1011” symbol for the asterisk symbol “*”. The group select output (GS) has to be held high when any key is pressed. The encoder also has an enable input (EI) and an enable output EO to detect when the encoder is disabled or when is active but no one is pressing keys. Finally, a 7-segment output will represent the code of the key pressed.

![Diagram of a PBX door phone with a keypad.](image)

The emphasis is set therefore in learning basic polling digital inputs and writing digital outputs. The truth table will be solved using a behavioural plan B interpretation in C language, organising the hardware-software diagram so that the input and outputs variables will be held in RAM memory. Your planning has to be similar to the one discussed in Chapter 1, but using C code instead of VHDL. The symbol for the $Enc_{12\_4}$ that will have priority high as usual in this kind of devices is represented in Fig. 68.

1. **Specifications.** Draw the truth table for this $Enc_{12\_4}$.
2. **Plan.** Propose a hardware-software diagram naming all the electrical signals, RAM variables and the software functions.
Hardware. Copy and adapt a circuit from any of the previous projects and name it Enc_12_4.pdsprj. Make the pin assignment connections accordingly to the options a, b or c given by your instructors.

Software. Explain how to configure the µC in init_system(). Organise using a flowchart the interface function read_inputs(). Organise using a flowchart the interface function write_outputs(). Infer the truth_table() software function using a behavioural interpretation and the corresponding flowchart.

3. Developing. Write the Enc_12_4.c. source code translating the function flowcharts.
   Start a software IDE project for the target microcontroller PIC18F4520 and generate the configuration files “.cof” and “.hex” after compilation. Discuss the project summary: % of ROM used for the code, number of RAM bytes used, etc. Find the RAM memory position of the variable Var_D. How many bytes does it occupy?

4. Test. Do it interactively in Proteus every time a few lines of code are added to the source file. Measure how long does it take to run the main loop code when using a 4 MHz oscillator.

P10 Programing FSM in C style. Events detection using interrupts

Objectives
After studying the content of this chapter on the basics of microcontrollers, you will be able to:
• Explain how a FSM can be organised in a microcontroller.
• Detect an events like rising or falling edges using interrupts.
• Draw the hardware-software diagram of an application based on FSM.
10.1 1-digit BCD counter

This is a plan X example to relate the design of FSM to Chapter 2.

A tutorial on this project is available [here](#).
10.2 Binary counter modulo 256

This is a plan Y example to relate the design of FSM to Chapter 2.

Project files are available [here](#).

10.3 4-bit serial data transmitter

Let’s design a simple 2-wire asynchronous data transmitter based on a µC for sending to another computer a nibble (4-bits) of data. It is basically a right-shift register. The application symbol and pinning of the PIC18F4520 is represented in Fig. 69. We’ll use the FSM style of programming in C language. The format for the serial output once the start-transmission ST rising edge is detected by means of an interrupt is: Start-bit ('0'), Data_in (0), Data_in (1), Data_in (2), Data_in (3); and then the end-of-transmission EoT pulse is generated to indicate that the transmitter has ended the process (see the Fig. 70). Serial_out is held high when idle.

![Fig. 69 Symbol of the data transmitter and the µC PIC18F4520 from Microchip.](image)

- a) Draw the hardware schematic. Reset circuit, XTAL oscillator, Data_in(3..0) = (RA2, RA1, RD7, RD6), CLK (RB0), ST (RB1), Serial_out (RC5), EoT (RC2). Explain how to configure the inputs and outputs in the init_system().
Fig. 70
Example of a section of a timing diagram where it can be seen how the data is read and right shifted in a single wire.

b) Draw the hardware/software diagram indicating the required RAM variables and how the FSM is solved in software. The transmission sequence will start when a rising edge is detected at the start ST push button by means of the interrupt INT1IF. The CLK input will generate an interrupt INTOIF so that a new bit is transmitted at a time at the Serial_out as represented in Fig. 70. Transmission speed is 150 b/s.

c) How read_input() works to generate the char variable var_Data_in?

d) How the variables var_Serial_out and var_EoT are written to the corresponding pins using write_outputs() without interfering the other μC port pins?

e) Which is the ISR() used for? Propose the flow chart.

f) Draw an state diagram showing the state transitions and the outputs for each state. Name the states, for instance: Idle, Start_bit, Data_0, Data_1, etc.

g) Draw the truth tables and their equivalent flow charts for the state_logic() and output_logic().

h) How to use and program the TMR0 peripheral in 8-bit mode to replace completely the functionality of the external CLK as the baud-rate generator?

A tutorial on the project can be read here.
10.4 5-bit Johnson counter

A tutorial on this project is available in these references (1), (2).
10.5 Stepper motor controller
P11 Peripherals: LCD display

Objectives
After studying the content of this chapter on the basics of microcontrollers, you will be able to:

- Explain ....
- ...

11.1 Basic interface for a LCD display

11.2 Interfacing an I2C display
P12 Peripherals and more complex applications

Objectives
After studying the content of this chapter on the basics of microcontrollers, you will be able to:

- Explain ....
- ...

12.1 *Industrial application*

This problem is connected with problem 8.3. The idea now is to design it using a microcontroller instead of a dedicated hardware design in VHDL. Read the assignment and solve the initial questions 1, 2, 3, 4.

Continue the problem as follows:

5. Draw the hardware schematic for an Atmel ATmega8535 microcontroller. Connect all the inputs and outputs to convenient I/O port pins, the reset button and the 12 MHz quartz crystal.

6. Architecture of the software. Organise and describe the program variables. Explain the use of interrupts. Assume that in a Phase #1 of the design, an external CLK signal of 4 Hz is available to generate the warm, hot and cold water timing periods of 50, 10 and 20 s respectively.

7. Describe the flowchart of bitwise operations for the functions to interface the hardware: `read_inputs()`, `write_outputs()` and `ISR(interrupt service routine)`. What kind of operations are solved by the `init_system()` function?

8. Describe the truth table and flowchart of the function to solve the state transitions: `state_logic()`.

9. Describe the truth table and flowchart of the function to implement the output variables: `output_logic()`.

10. Explain how to implement the timing signal of 4 Hz s internally using the Timer0 counter/timer peripheral in a Phase #2 of the project.
12.2 Simple remote control

We want to design a very simple wireless infrared remote control for an electronic equipment as shown in Fig. 71. In this initial stage of the design, the Chip1 is the microcontroller while the other components are external integrated circuits. The Chip2 (decoder BCD to 7-segments) is used to show the channel number, the Chip3 is the infrared transmitter, and the Chip4 is a 2 seconds CLK. Furthermore, the volume control is not implemented and thus only the buttons BU and BD are considered.

The system has a capacity of 7 channels. To increment the channel the BU (Channel up) has to be pressed. To decrement the channel number the BD (Channel down) has to be pressed. If both buttons are pressed or released simultaneously, the channel count is maintained. The buttons are sampled every 2 seconds (0.5 Hz). The outputs C(2..0) represents the channel selected in binary.

Design phase #1
1. Timing diagram.
2. State diagram. The initial state is the Channel 1.
3. Hardware circuit. Connect input and output pins to the microcontroller ATmega8535, a master reset and crystal oscillator of 8 MHz.
5. Functions to interface the hardware: init_system(), read_inputs(), write_outputs(), ISR(source of interrupt). Flowchart of bitwise operations.
6. The function to implement state transitions: state_logic(). Truth table and flowchart.
7. The function to implement the outputs: output_logic(). Truth table and flowchart.

Design phase #2
8. If we like to include into the microcontroller the functionality of the Chip2 (decoder BCD to 7-segments), so that this external chip will be no longer required, how to proceed?
9. If we like to include the Power_ON/OFF button into the microcontroller, so that when clicked the code “000” is generated immediately, how to proceed?
12.3 Non-retriggerable timer

1. Specifications
Our aim is to design a timer of exactly 11.25 s as represented in Fig. 72. It is non-retriggerable, which means that the system is not affected even if you click the Trigger more than once while active in the timing period. In this project, the strategy will be to count external/internal pulses once the trigger signal is detected as represented in the timing diagram. The project is based on a PIC18F4520 microcontroller from Microchip. We will consider two design options and you have to choose one of them:

**Option A:** Using the external 16 Hz clock input as the INT1 interrupt source.

**Option B:** Using the internal TMR0 peripheral instead of the 16 Hz clock input.

2. Planning
A microcontroller-based architecture running a FSM.

1. Hardware: draw the schematic indicating where to connect and how the system oscillator, the reset circuit, and all the remaining inputs and outputs.

2. Software: Draw a possible state diagram for the timer system. How many states will this FSM contain? Which is the task to be performed in each state?
3. Infer all the software variables (names and types) that will be required for managing the application. How many bytes of RAM memory will require? What kind of variable is current_state?

4. Explain how to organise the software (main, setup, interrupts, write outputs, etc.) and describe the operations to setup the system init_system() and explain how to set a pin as input or output.

5. Describe the bitwise operations and the flow chart required to write the output: write_outputs().

6. Write the C code lines of the interrupt service routine ISR().

7. Describe the truth table and how to organise the flow chart of the state_logic() function.

8. Describe the truth table and how to organise the flow chart of the output_logic() function.

3. Development and D. Verification

9. Which EDA and debugging tools and techniques are you going to use to compile the code and test the system in Proteus?

10. (extra) Download the microcontroller’s configuration file to the PICDEM 2 Plus board and verify how it works connecting the LED at the T_out output. If the Option A was chosen, use the laboratory’s signal generator to obtain the 16 Hz square wave that has to be applied at pin RB1/INT1.
12.4 Timers. PWM generation

Fig. 73 shows the symbol of an application based on the PIC18F4520 (Fig. 74) running with an 8 MHz crystal quartz oscillator. The idea is to control the rotation speed of a direct current motor generating a 25 Hz waveform that has 2 possible selectable duty cycles: DC1 = 20% and DC2 = 80%. The 7-segment display will show the sign “-“ when idle, and the numbers 1 or 2 depending on the DC selected by the switch. The button B starts and stops de waveform generation. Fig. 75 shows an idea of the state diagram.

![Diagram of the application](image)

a) Draw the two waveforms indicating the $T_{ON1}$, $T_{OFF1}$, $T_{ON2}$ and $T_{OFF2}$ periods of time.

b) Draw the schematic connecting the inputs and outputs to the PIC18F4520. Add the crystal oscillator and the MCLR_L circuits. Explain how to configure the inputs and outputs in the `init_system()`.

c) Explain how to connect and configure the TMR0 (Timer 0) peripheral to generate interrupts. Which are the necessary N1 and N2 values for the pre-scaler and the TMR0 counter to be able to generate all the
required timing periods?  

\[ Timing\_period = \left(\frac{4}{F_{\text{osc}}}\right) \cdot N1 \cdot N2 \]

d) Draw the hardware/software diagram indicating the required RAM variables and how the FSM is solved in software. How to implement the functions read_inputs(), write_outputs() and ISR()? How and where to drive the 7-segment display to show the sign “-“ and the numbers “1” and “2”?

e) Complete the state diagram represented in Fig. 75 and deduce the truth tables for the main functions of the C code: state_logic() and output_logic().

Problem discussion.

12.5 Temperature measurement using timers

12.6 Temperature measurement using A/D converters
Bibliography and internet links

Bibliography


Internet links

[8] This is what is expected from students and instructors.
[9] CSD instructions for professional communications by email.
[10] Thunderbird e-mail client.
[12] Dropbox cloud file system
[16] 
[17]
Examples of questions

P1

Examples of questions for P1 and P2.

P2

(Let’s add here examples of questions from P1 to P12 so that they can be used to study the subject)
This list has to contain all the CSD topics

Behavioural design approach, 22
Encoders, 22
  group select (GS), 22, 28
  priority high, 22
Flat design approach, 22
Product-of-sums (PoS), 22

Standard combinational circuits
  74HC147 - 10 to 4 line priority encoder, 27
Sum-of-products (SoP), 22
Truth table, 22