



Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels



Problems on digital circuits and systems (CSD)

This is the former CSD problem collection. Not currently updated, kept only as a reference for old digsys links.

The idea is to transfer and rewrite problems from P1 to P12 to digsys web pages as sample projects.

Francesc J. Robert
Josep Jordana







Contents

Preface	11
Combinational circuits.....	15
P1 Logic gates and Boolean algebra.....	16
Objectives.....	16
1.1 Circuit analysis, truth tables and maxterms and minterms	17
1.2 Design Circuit_C using minimised equations	20
1.3 Design Circuit_K using minimised equations	21
1.4 Circuits using only NOR or only NAND	22
1.5 Circuits using only NOR or only NAND	23
1.6 Logic equations (PoS, SoP, maxterms, minterms).....	23
P2 Standard logic circuits and flat VHDL design.....	25
Objectives.....	25
2.1 Using VHDL EDA tools for synthesis and simulation	26
2.2 Designing a MUX_8 using several architectures	27
2.3 Designing a HEX_7SEG_decoder	28
2.4 Designing a 10-line to 4-line priority encoder.....	29
2.5 Designing an 8-line to 3-line priority encoder	32
2.6 A digital wind direction meter	34
P3 Arithmetic circuits: adders, multipliers, comparators, etc. and VHDL hierarchical design (plan C2)	39
Objectives.....	39
3.1 Logic functions using the methods of decoders	40
3.2 Logic functions using the method of multiplexers.....	41
3.3 Design a 1-bit full adder (flat)	42
3.4 Design a 1-bit full adder (structural)	43
3.5 Design a 1-bit comparator	44
3.6 Designing a MUX_8 using a multiple-file structure.....	45
3.7 4-bit ripple adder	46
3.8 8-bit binary adder using 4-bit carry-look ahead adders.....	47
3.9 Designing a 6-bit comparator using VHDL	48
3.10 Counting occupied parking slots (32-bit ones' counter).....	50
3.11 1-bit subtractor	52
P4 Arithmetic circuits for 2C integer numbers and gate-level simulations for propagation delay measurements	53



Objectives.....	53
4.1 Addition and subtraction in two's complement	54
4.2 Designing an 8-bit adder/subtractor for integer numbers	55
4.3 Designing a 10-bit comparator for radix-2 and 2C numbers	56
4.4 Performing gate-level simulations and propagation time measurements.....	57
4.5 How to design an 8-bit multiplier for 2C integer numbers?	58
Sequential systems.....	61
P5 1-bit memory cells: latches and flip-flops.....	62
Objectives.....	62
5.1 Designing and using an RS latch. Deducing an RS_FF.	63
5.2 Data flip-flop (D_FF).....	64
5.3 Analysis of a synchronous circuit	65
5.4 JK_FF and analysis of an asynchronous circuit.....	66
5.5 Analysis of a synchronous circuit	69
5.6 Design a toggle flip-flop (T_FF) using the FSM strategy.....	70
5.7 Design a JK flip-flop using the FSM strategy	71
5.8 Analysis of an asynchronous counter (type 7493).....	72
5.9 Analysis of an asynchronous circuit based on T_FF.....	73
5.10 Design a combinational circuit using the method of ROM memories.....	74
5.11 Design a HEX_7seg using the method of ROM memories.....	75
P6 Finite State Machines (FSM)	76
Objectives.....	76
6.1 Controlling the classroom luminaires	77
6.2 Invent a bicycle torch	78
6.3 Debouncing circuit	78
6.4 16-key matrix encoder	79
6.5 Water tank controller	80
6.6 Traffic light controller	82
6.7 Stepper motor controller	83
6.8 7-segment digit sequencer	84
P7 Standard counters and registers.....	86
Objectives.....	86
7.1 1-digit BCD counter (flat)	87
7.2 Synchronous universal 4-bit binary counter	88
7.3 Synchronous modulo 12 counter	90
7.4 Data register.....	91
7.5 Shift register.....	92
7.6 Hour counter for a real-time clock.....	93



7.7	6-bit binary universal counter.....	95
7.8	Johnson counter.....	97
7.9	PIC18F4520 TMR2 prescaler design.....	99
P8 Dedicated processors and advanced circuits		101
Objectives.....		101
8.1	Generation of CLK signals.....	102
8.2	Pulse generator	103
8.3	Designing an industrial application	105
8.4	Design a 2-digit even/odd counter with start/stop button	107
8.5	Synchronous serial adder	108
8.6	Timer MMSS.....	109
8.7	Synchronous serial multiplier.....	110
8.8	Serial transmitter and receiver (USART)	111
8.9	Steeping motor control based on a dedicated processor.....	112
Microcontroller applications		114
P9 Basic theory on microcontrollers (μC) and basic digital I/O interface		114
Objectives.....		114
9.1	The microcontroller PIC16F.....	115
9.2	Invent a Dual_MUX4 based on a μ C	118
9.3	1-digit BCD adder	119
9.4	12-to-4 encoder	121
P10 Programing FSM in C style. Events detection using interrupts.....		123
Objectives.....		123
10.1	1-digit BCD counter.....	124
10.2	Binary counter modulo 256	124
10.3	4-bit serial data transmitter.....	125
10.4	5-bit Johnson counter	127
10.5	Stepper motor controller.....	128
P11 Peripherals: LCD display		131
Objectives.....		131
11.1	LCD display using ASCII messages and static data	132
11.2	LCD display using dynamic data.....	132
11.3	Interfacing an I2C display.....	132
P12 Peripherals and complex applications		133
Objectives.....		133
12.1	Industrial application	134
12.2	Simple remote control	135
12.3	Non-retriggerable timer.....	137
12.4	Timers. PWM generation.....	139



12.5	Temperature measurement using timers.....	141
12.6	Temperature measurement using A/D converters	141
Bibliography and internet links.....		143
Bibliography		143
Internet links		143



Preface

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](#).

Home Term 1920Q1 Contact Electronic devices and companies Software Books Instruments Magazines RPi Arduino Search

Digital Circuits and Systems (CSD)

Content and skills

Chapter 1	Chapter 2	Chapter 3
Combinational Circuits	Sequential Systems	Microcontrollers
VHDL		C language
EDA tools for sPLD/CPLD/FPGA. Synthesis and simulation: Lattice Semiconductor ispLEVER Classic/Diamond/Active-HDL, Intel Quartus II/ModelSim Starter or Xilinx ISE/ISim		EDA tools for microcontrollers: Microchip (PIC18F/16F/Atmega) MPLABX/XC8
		P12 - P_Ch3 : Timers, A/D, I2C, USART, etc.
		P11 : Peripherals: LCD display
		P10 : FSM style of programming. External Interrupts
		P9 : Microcontroller architecture. Digital I/O
	P8 - P_Ch2 : CLK generators and dedicated processors	
	P7 : Counters, data and shift registers	
	P6 : Finite State machines (FSM)	
	P5 : 1-bit memory cells: latches and flip flops	
	P4 - P_Ch1 : Arithmetic circuits and ALU : two's complement (2C) operations	
	P3 : Arithmetic circuits (radix-2): adders, comparators, etc. Multiple-file structural VHDL design	
	P2 : Standard logic circuits: Multiplexers, demultiplexers, decoders, encoders. Single-file VHDL design	
	P1 : Logic gates and Boolean Algebra (SoP, PoS, maxterms, minterms, schematics, truth tables, etc.)	
Cross-curricular competences (1) (2) (3) (4) (5)		

Course organisation and basic learning goals

- Use your official UPC email address to communicate with your instructors for these reasons [8].
- Use and manage an e-mail client like Thunderbird [9] 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](#) and publish your projects, results and reflection.

→ 1



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).

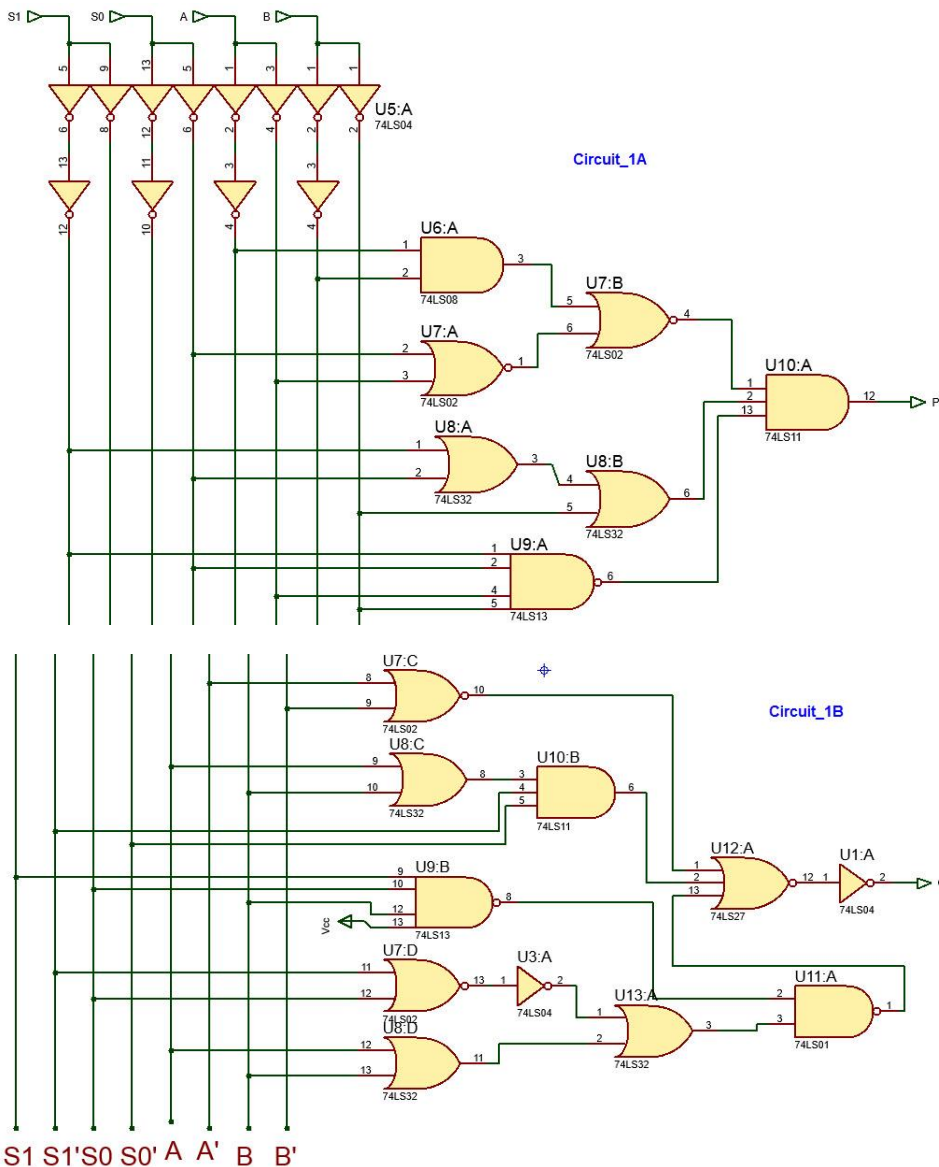


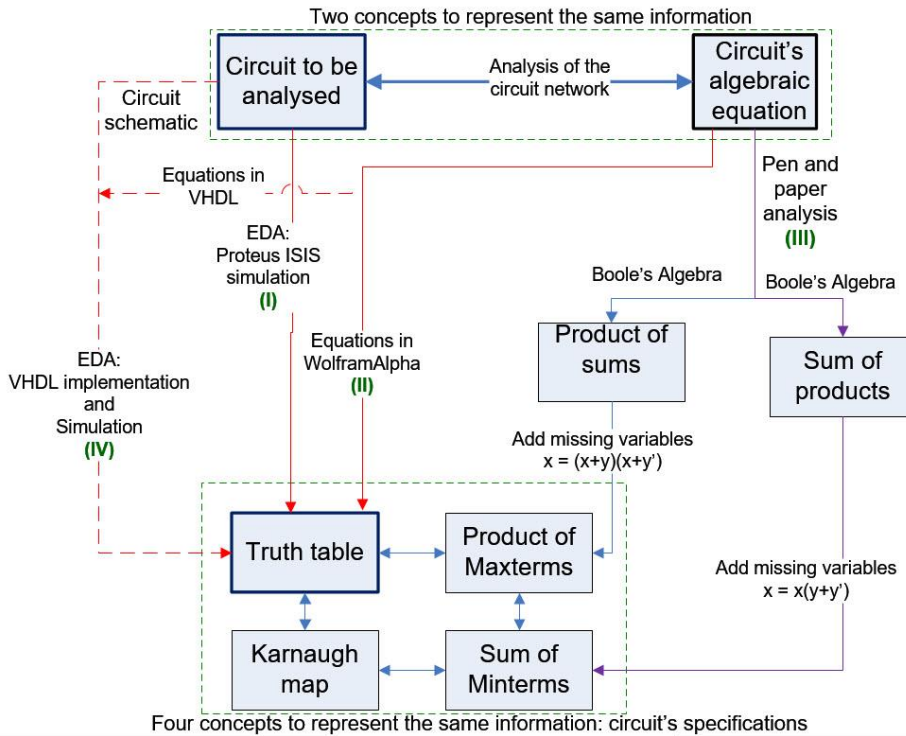
Fig. 1
Circuit_1A and
Circuit_1B composed
of a network of logic
gates.

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.

Fig. 2
Multiple planning
paths to analyse
a circuit 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.

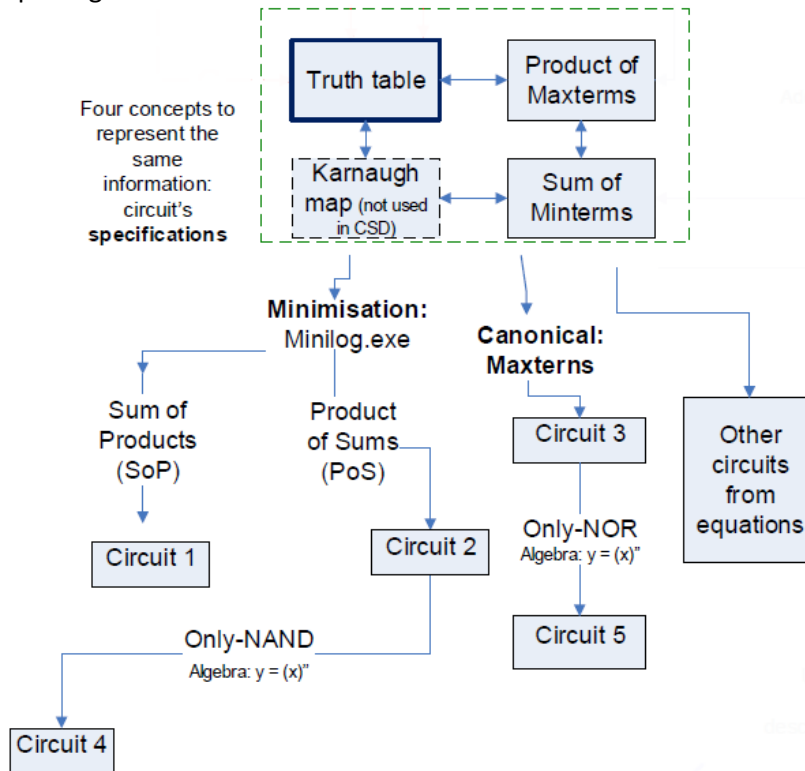


Fig. 3
Invention of circuits from the same initial truth table.

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.

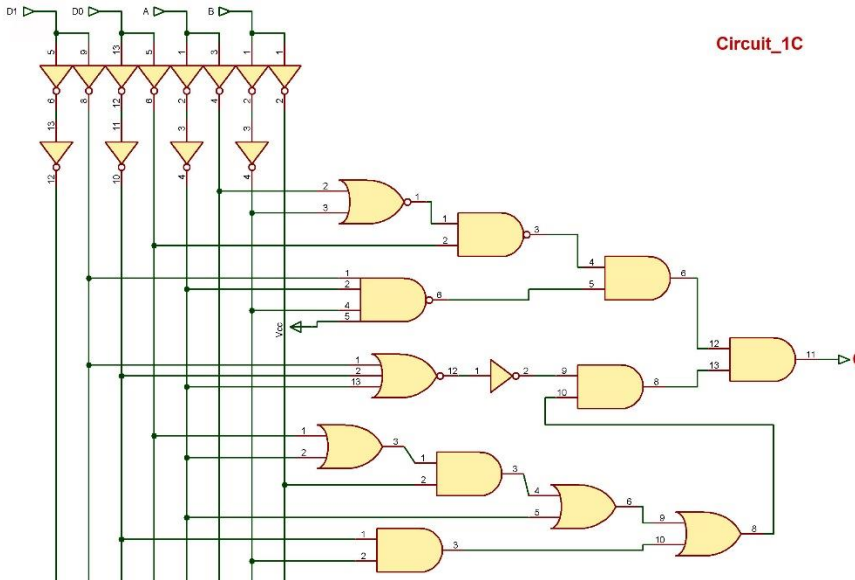
✚ Some ideas on the solution to the analysis problem can be found in this [tutorial](#).



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).

Fig. 4
Circuit_C
composed of a
network of logic
gates.



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).

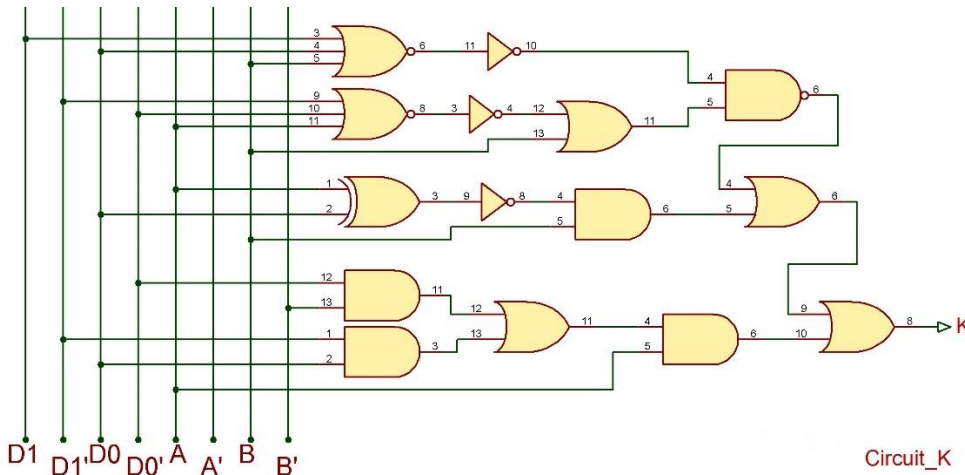


Fig. 5
Circuit_K composed
of a network of logic
gates.

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

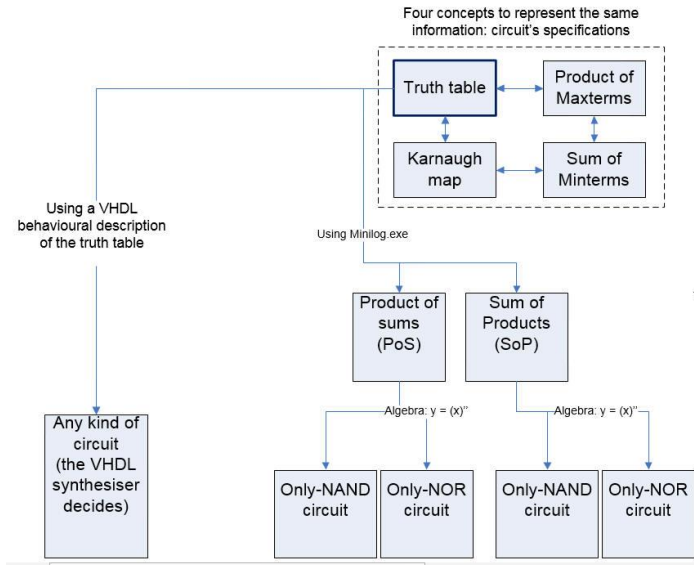
Given the following Boolean expression:

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

- 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_{PHL} and t_{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.

Fig. 6
Concept map to explain the idea of extracting several circuits from the same initial 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

```
=====
FOUND 28 ESSENTIAL FACTORS IN
PRODUCT OF SUMS MODE
MAXIMUM FANIN:          18
TOTAL LITERAL COUNT:    102
MAXIMUM FACTOR SIZE:    3
MAXIMUM OUTPUT FUNCTION SIZE: 6
```

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

```
=====
                ABE
                LLL
=====
DCBA  |
=====
011-  | 1..
10-0  | 1..
-1-1  | 111
0--1  | 1..
1-0-  | 1..
-00-  | 1..
11--  | .1.
0-10  | .1.
1-0-  | .1.
1-11  | .1.
```

Fig. 7
Output table format
from a given circuit
described in Minilog.

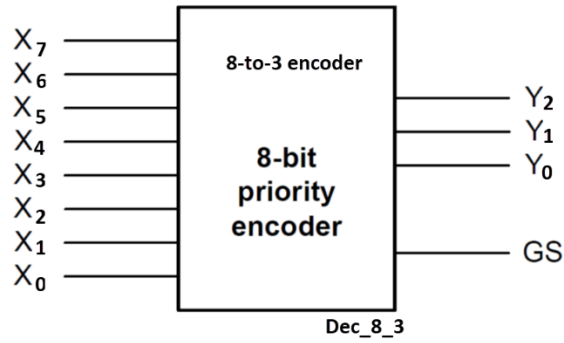
1.6 Logic equations (PoS, SoP, maxterms, minterms)

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 $Y_2 = f(X_7 \dots X_0)$ using a product of sums (PoS).
- Represent the output $Y_1 = f(X_7 \dots X_0)$ 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.



Fig. 8
The symbol and
truth table of a
combinational
circuit.



X_7	X_6	X_5	X_4	X_3	X_2	X_1	X_0	Y_2	Y_1	Y_0	GS
0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	0	0	0	1	-	0	0	1	1
0	0	0	0	0	1	-	-	0	1	0	1
0	0	0	0	1	-	-	-	0	1	1	1
0	0	0	1	-	-	-	-	1	0	0	1
0	0	1	-	-	-	-	-	1	0	1	1
0	1	-	-	-	-	-	-	1	1	0	1
1	-	-	-	-	-	-	-	1	1	1	1



P2 Standard logic circuits and flat VHDL design

Objectives

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

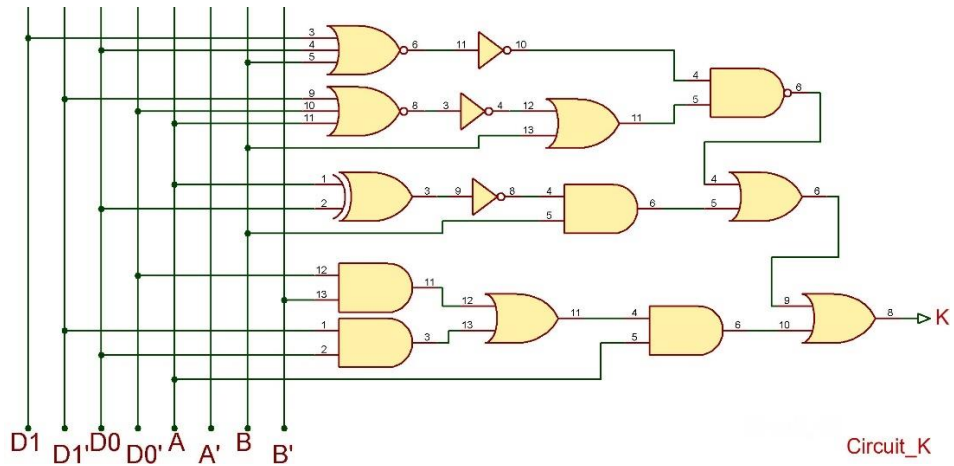
- Explain the specifications and characteristics of the standard combinational logic blocks: multiplexers (or data selectors), demultiplexers (or data distributors), decoders, encoders, hexadecimal to seven-segment LED displays adapters, code converters, etc. Specifications include concepts like: symbol, 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.

Fig. 9
Example of a
combination circuit
to be analysed.



- Understand the specifications and characteristics of VHDL synthesis and simulation tools.
- Organise a plan detailing the sequence of operations to reach the end of the problem successfully.
- Develop the solution.
- 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 file¹ 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 chip's functionality.
- Plan C1. Invent a hierarchical schematic for MUX_8 using smaller blocks of the same kind, for instance, MUX_4 or MUX_2.

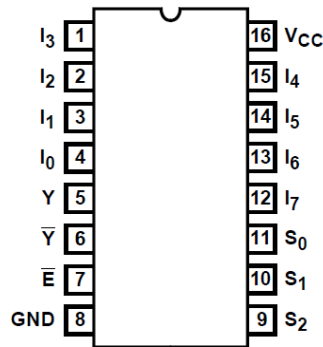
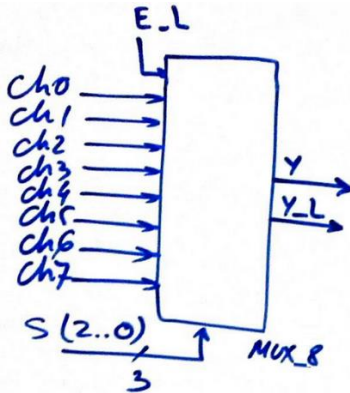



Fig. 10
MUX_8 chip symbol
derived from the
standard 74HCT151.

 Problem [discussion](#).

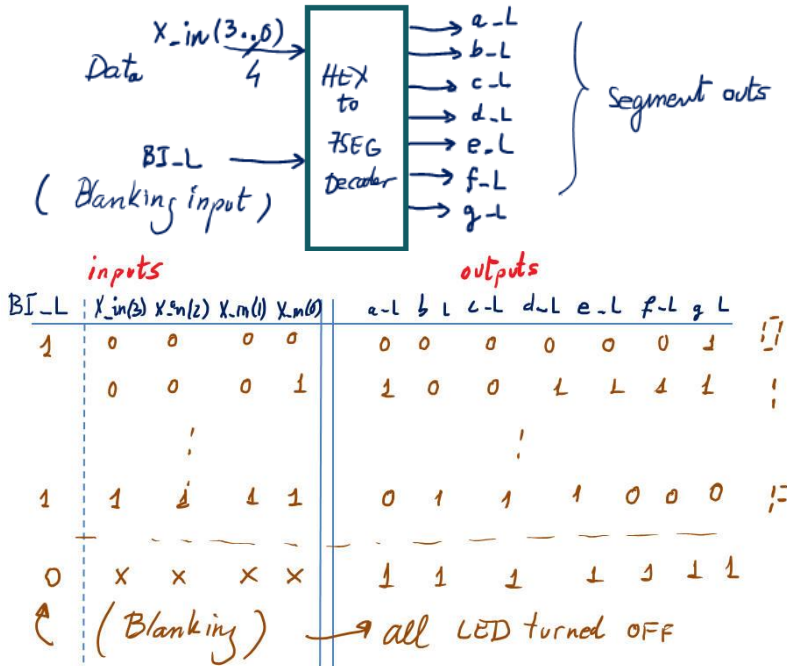
¹ The plan C2 proposed in P3 can be used to design hierarchical structures using multiple-file projects, which enables the design of complex, large circuits.



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. 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.

Fig. 11 Hexadecimal to seven-segment decoder



- Plan A. Structural, using logic equations.

+ Problem [discussion](#).

- Plan B. Behavioural, using the truth table or a high-level algorithm describing the chip's functionality.

+ Problem [discussion](#).

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

The objective is to develop VHDL code and the final circuit for the 10-line to 4-line priority 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 priority decoding of the inputs to ensure that only the highest-order data line is encoded 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*.

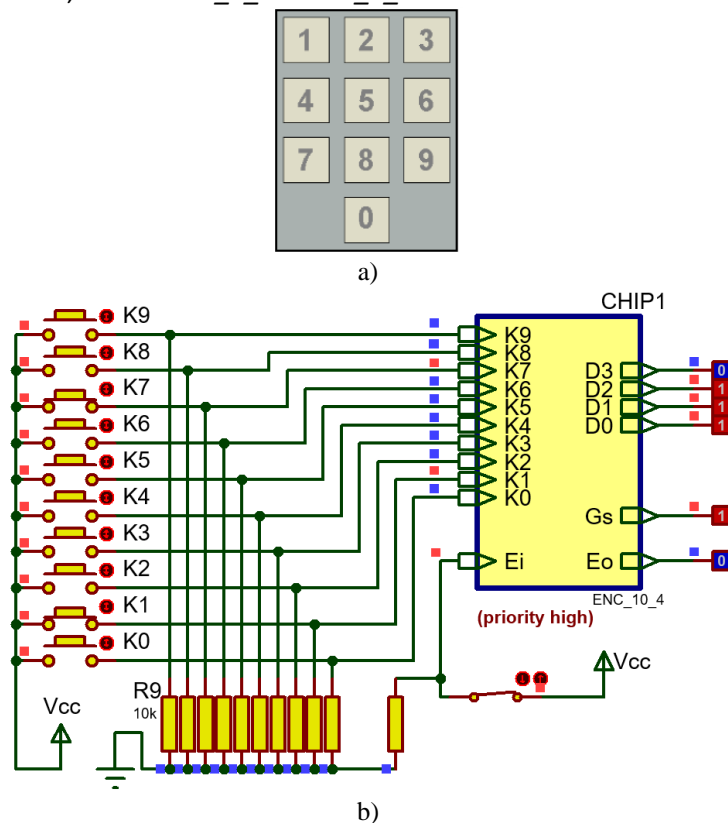


Fig. 12

a) A typical 10-key numerical keypad (push-buttons).

b) Interface circuit that gives the binary code of the pressed key. This is an MSI - (medium scale of integration) circuit that can be fitted in a simple programmable logic device (sPLD) like the [GAL22V10](#) or any other CPLD or FPGA.

The picture shows the outputs when the circuit is enabled and the keys K7 and K1 are pressed simultaneously.

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. **Gs** has to be high (‘1’) when any key is pressed. The enabled output (**Eo**) is high when the circuit is enabled (**Ei** = ‘1’) and the keys are not pressed. How many combinations does the truth table include?

Ei	K9	K8	K7	K6	K5	K4	K3	K2	K1	K0	D3	D2	D1	D0	Gs	Eo	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 $Min_Pulse = 10 \mu s$?
- 5) How do you generate a ‘0’ or a ‘1’ using push-buttons or switches? How do you drive a LED for example to connect at the output **Gs**? Study [this circuit](#) in Proteus to get ideas and design formulas.
- 6) How many minterms will **D3** have? How many maxterms will **D2** 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 **D1** using only NOR gates.

$$(x \cdot y)' = x' + y' \quad , \quad (x \cdot y \cdot z)'' = (x' + y' + z')'$$
- 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 δ ?
- 10) Prepare other similar questions such as: invent **D0** 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 *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 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.

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.

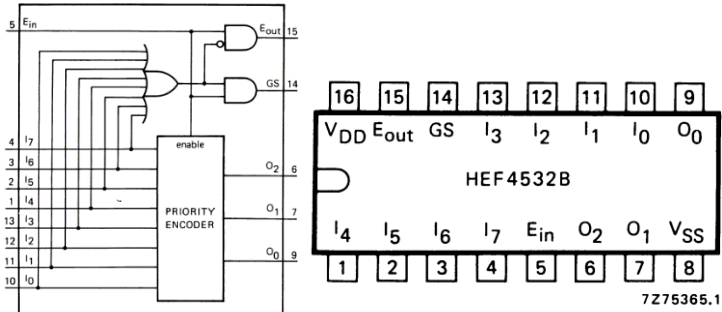
Looking forward: this project introduces standard combinational encoder devices. They can be compared to matrix implementations such in Problem 6.4.



2.5 Designing an 8-line to 3-line priority encoder

Fig. 13 shows the internal circuit of the classic chip [HEF4532B](#) and its truth table as specified by Philips.

Fig. 13
The HEF4532B manufactured by Philips. When E_{in} is low the chip is disabled. Group select (GS) is assessed when there is any input active. Enable output E_{out} is active only when the chip is enabled and there is not a single input active. The outputs $O(2..0)$ generate the binary code of the input being active. The highest input prevails when more than one input is active at the same time.



INPUTS									OUTPUTS				
E_{in}	I_7	I_6	I_5	I_4	I_3	I_2	I_1	I_0	GS	O_2	O_1	O_0	E_{out}
L	X	X	X	X	X	X	X	X	L	L	L	L	L
H	L	L	L	L	L	L	L	L	L	L	L	L	H
H	H	X	X	X	X	X	X	X	H	H	H	H	L
H	H	L	H	X	X	X	X	X	H	H	H	L	L
H	H	L	L	H	X	X	X	X	H	H	L	H	L
H	H	L	L	L	H	X	X	X	H	H	L	L	L
H	L	L	L	L	H	X	X	X	H	L	H	H	L
H	L	L	L	L	L	H	X	X	H	L	H	L	L
H	L	L	L	L	L	L	H	X	H	L	L	H	L
H	L	L	L	L	L	L	L	H	H	L	L	L	L

- 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?
- 2) Write $E_{out} = f(E_{in}, I_7, \dots, I_0)$ using minterms. In addition, draw the equivalent circuit using gates.
- 3) Write GS as a product of sums (PoS). How many minterms does this function have? Draw the circuit using gates.
Write O_2 as a sum of products (SoP). How many maxterms does this function have? Write O_1 and O_0 as a product of sums (PoS).
- 4) Describe the circuit in VHDL in a behavioural or structural fashion.
- 5) Draw a sketch of a timing diagram and translate it into a test bench so that the circuit can be verified using an EDA tool.

Optional questions related to further understanding the problem and designing the project *encoder_8_to_3.vhd* into a programmable device:

- 6) Write the *.tbl* format file so that it can be used to obtain the PoS or the SoP in Minilog software. Find and write down the link to a similar circuit in



[HADES](#) that can be executed using the Java applet. Simulate the circuit in Proteus and check if it works as expected.

- 7) Start a VHDL-based synthesis project and a testbench-based simulation using EDA tools for target CPLD or FPGA programmable chips.



2.6 A digital wind direction meter

We want to design a digital wind direction meter (*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:

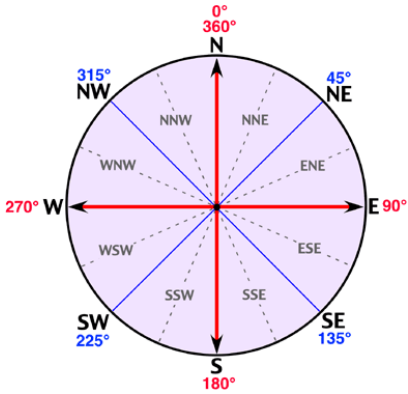
- Plan A: Structural (flat design with a single VHDL file), using logic equations in a canonical or simplified version.
- Plan B: Behavioural (flat design with a single VHDL file), using high-level description of the specifications.
- Plan C2: Structural (hierarchical design with multiple VHDL files),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

Fig. 16 shows the symbol of the *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.

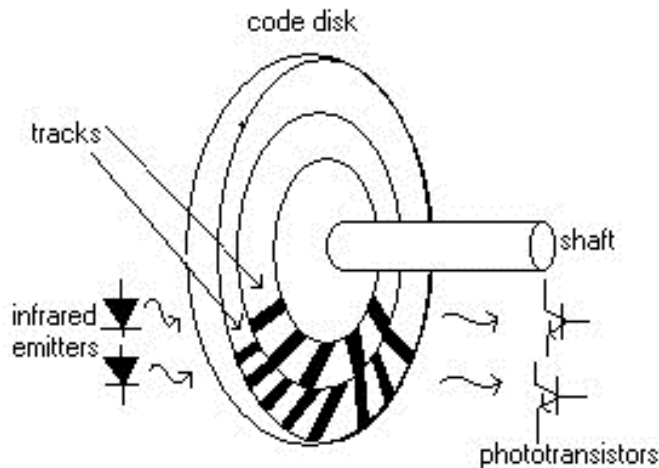
Fig. 14
a) Wind compass describing the sixteen principal bearings used to measure wind direction, b) the wind [transducer](#).



- 1) Write the truth table of the *wind_compass*. The inputs have to be ordered in this way: E, D(3..0).
- 2) Write the functions **Y(7)** and **Y(14)** canonically as a product of maxterms.



Fig. 15
The basics of the Gray
to binary rotary
[encoder](#) sensor.



- 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 equation 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.

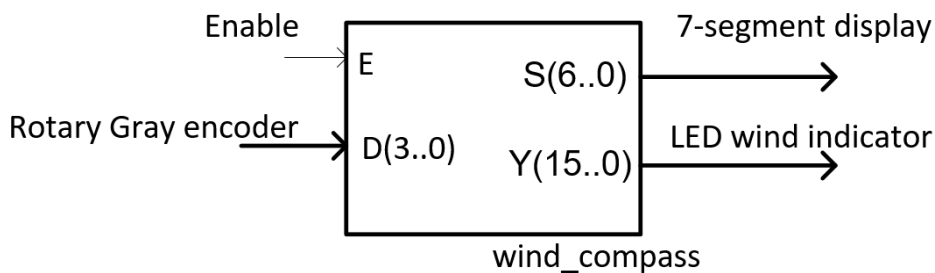


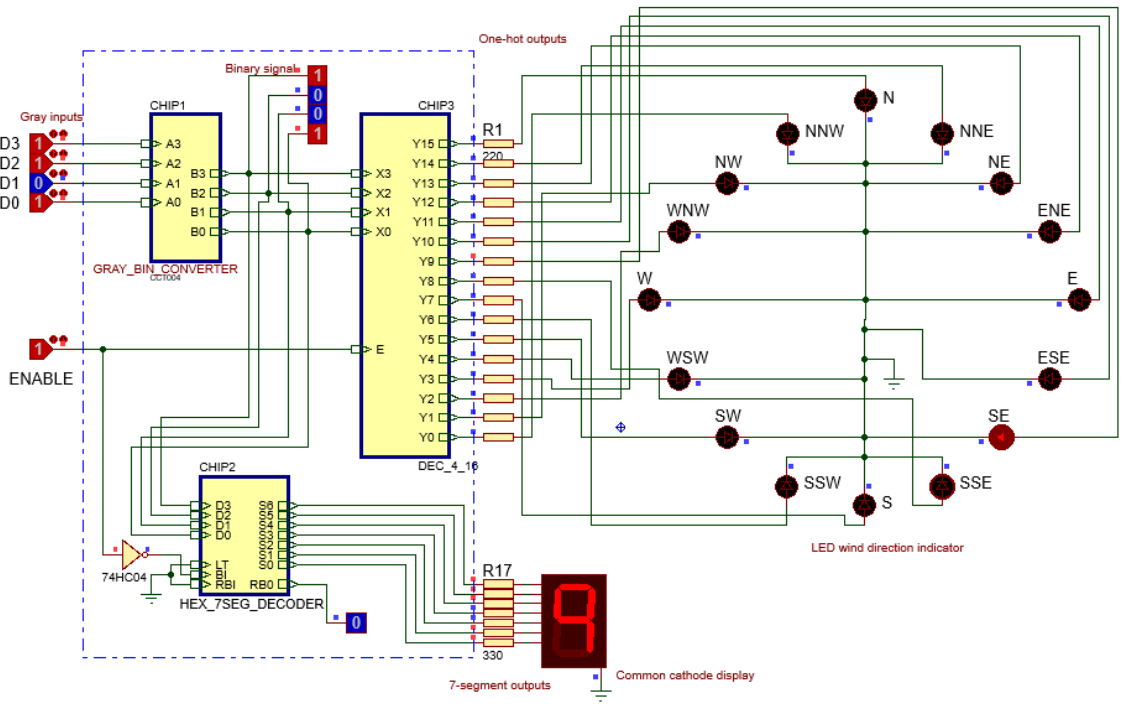
Fig. 16
Symbol of the project
wind_compass
representing and
naming all the inputs
and outputs.
Anyone of the 23
outputs is a function
of the inputs E, D(3),
D(2), D(1), D(0) in this
order, for instance:

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



Fig. 17 shows an example of an internal electronic schematic for the *wind_compass* when plan C2 is used. It is available [here](#) and for experimentation. The output of Chip1 (*gray_bin_converter*) is connected to both Chip2, a 1-digit 7-segment decoder (*hex_7seg_decoder*), and Chip 3, 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 $Min_Pulse = 1.26\ \mu s$, 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:\CSD\P2\wind_compassA\wind_compass.vhd
```



- 13) Plan B: write the VHDL code for the *wind_compass* to obtain the *wind_compass.vhd* circuit file.

L:\CSD\P2\wind_compassB\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. 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:\CSD\P3\wind_compass\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*.



Fig. 18

- a) Truth table of a 4-bit Gray to binary converter circuit.
 b) The translation of the truth table into Minilog text format.

Note how the 16 input combinations do not have to be written necessarily in binary sequential.

A3	A2	A1	A0	B3	B2	B1	B0
0	0	0	0	0	0	0	0
0	0	0	1	0	0	0	1
0	0	1	1	0	0	1	0
0	0	1	0	0	0	1	1
0	1	1	0	0	1	0	0
0	1	1	1	0	1	0	1
0	1	0	1	0	1	1	0
0	1	0	0	0	1	1	1
1	1	0	0	1	0	0	0
1	1	0	1	1	0	0	1
1	1	1	1	1	0	1	0
1	1	1	0	1	0	1	1
1	0	1	0	1	1	0	0
1	0	1	1	1	1	0	1
1	0	0	1	1	1	1	0
1	0	0	0	1	1	1	1

a)

table Gray_Bin_Converter

input A3 A2 A1 A0
 output B3 B2 B1 B0

```

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

end

b)



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: **corrected**

- Explain how to perform basic operations like add, compare or multiply using the 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 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 methods of decoders

The following function is expressed in SoP: [Here](#)

$$f(w, x, y, z) = x'y'z + x'y'z' + wxy' + wyz' + xy$$

- 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.
- 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.



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 using a MUX4. 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.



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:

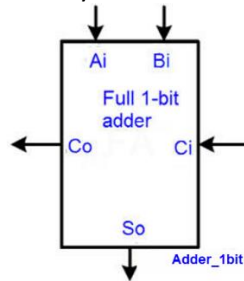
- A) Structural
- B) Behavioural

+ 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.

Fig. 19
The entity of an
Adder_1bit.





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.

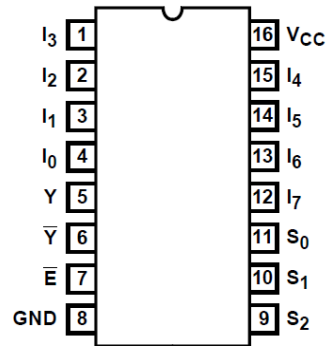
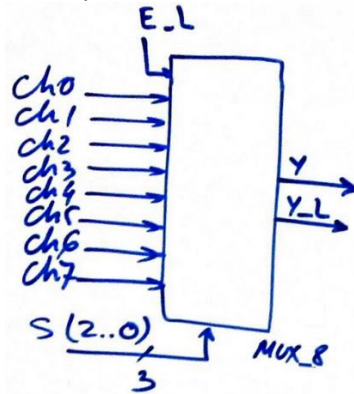



Fig. 20
MUX_8 chip symbol
derived from the
standard 74HCT151.

+ Problem [discussion](#)



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-look ahead adders

Project:

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

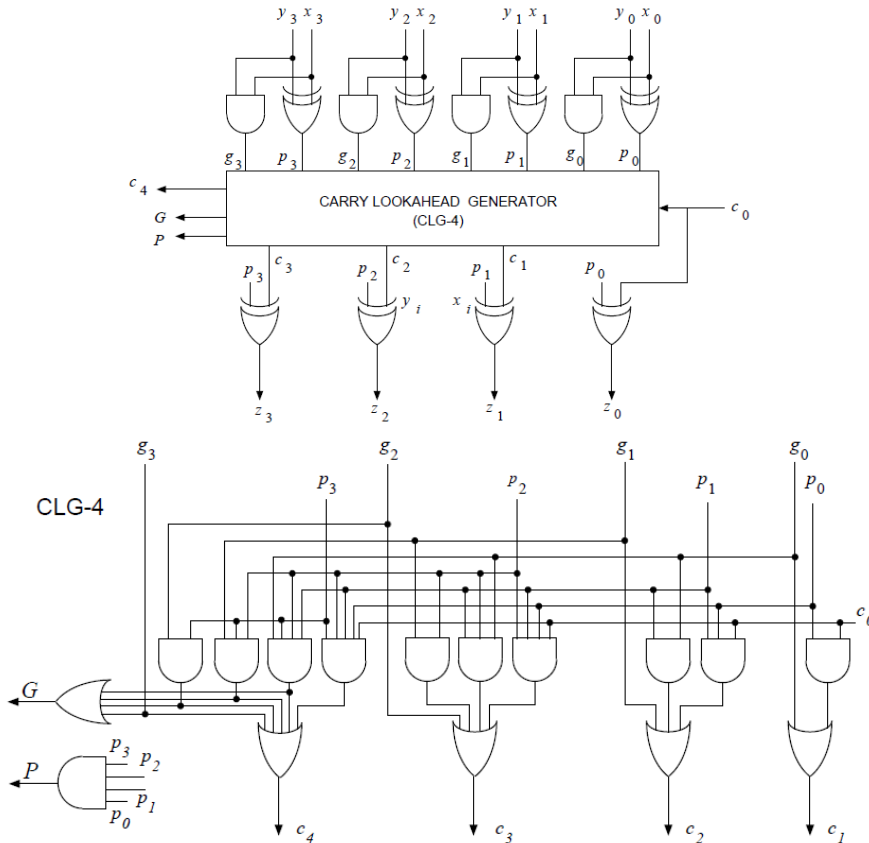


Fig. 21
The idea of a 4-bit carry-lookahead adder (chip [74HCT283](#)), as explained in: Introduction to digital systems, Ercegovic, M., Lang, T., Moreno, J. H., John Wiley & Sons, 1999, [web](#).

Development of the 4-bit carry-lookahead adder.

- 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.

Fig. 22
 a) Entity for the 6-bit expandable comparator.
 b) Example of an internal architecture for the *Comp_6bit* based on a structure of smaller elements of the same kind (Plan C2).

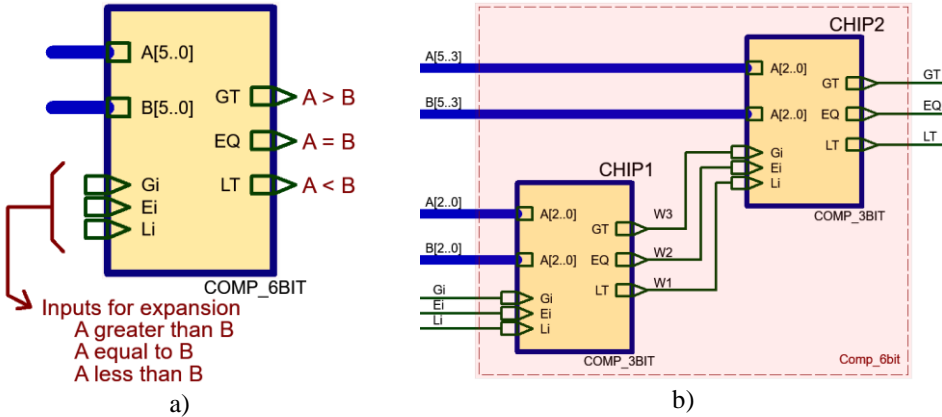


Fig. 23
 Truth table for the cascadable 3-bit comparator *Comp_3bit* which is used as a building component in the *Comp_6bit*.

<i>A</i> [2..0]	<i>B</i> [2..0]	<i>Gi</i>	<i>Ei</i>	<i>Li</i>	<i>GT</i>	<i>EQ</i>	<i>LT</i>	
A > B					1	0	0	A is greater than B
A < B					0	0	1	A is lesser than B
A = B					1	0	0	<i>Gi</i> input decides
					0	1	0	<i>Ei</i> input decides
					0	0	1	<i>Li</i> input decides

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

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



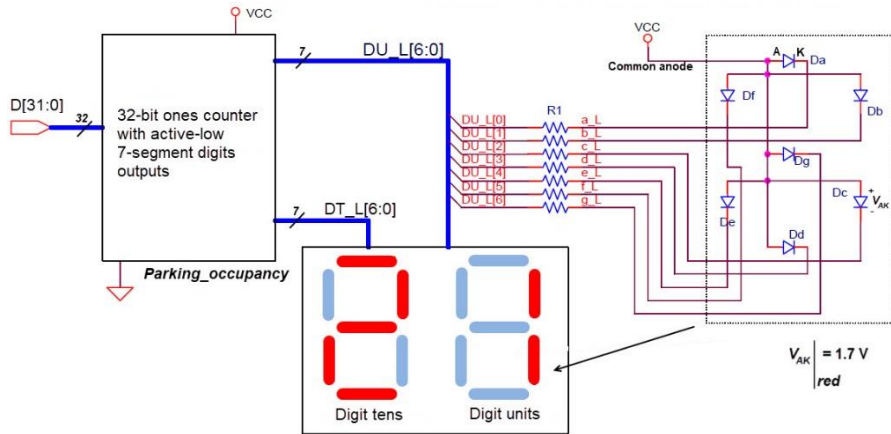
3.10 Counting occupied parking slots (32-bit ones' counter)

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 $D = "1001001111011111000111110101110"$ 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



- Draw and explain the internal architecture of the parking occupancy circuit based on components and representing some examples of the components truth tables.
- 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 **R1** in the worst case scenario if each segment must be biased with 15 mA when lighting.

Symbol	Parameter	Limits			Unit
		Min	Typ	Max	
V_{IH}	Input HIGH Voltage	2.0			V
V_{IL}	Input LOW Voltage			0.8	V
V_{OH}	Output HIGH Voltage	2.7	3.5		V
V_{OL}	Output LOW Voltage		0.25	0.4	V

Symbol	Parameter	Min	Max	Unit
t_{PLH}	Propagation Delay Time LOW-to-HIGH Level Output	4	15	ns
t_{PHL}	Propagation Delay Time HIGH-to-LOW Level Output			

- 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.

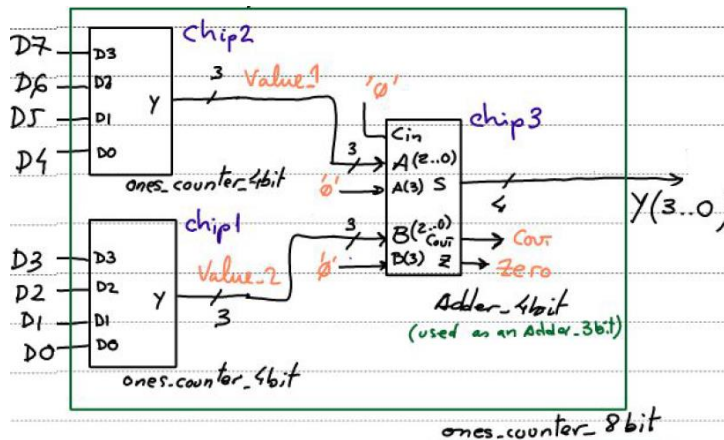


Fig. 25
 a) Example of building a *ones_counter_8bit* using smaller components like *ones_counter_4bit* and *Adder_4bit*.
 b) This is the truth table of a *ones_counter_4bit*

a)

D (3..0)	Y (2..0)
0 0 0 0	0 0 0
⋮	⋮
0 0 1 1	0 1 0
⋮	⋮
1 1 1 0	0 1 1
⋮	⋮
1 1 1 1	1 0 0

b)

+ Problem [discussion](#) including the 8-bit and the 4-bit ones' counter and a commented [solution](#).

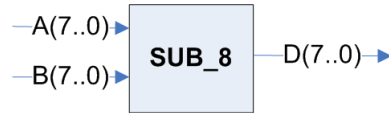


3.11 1-bit subtractor

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

Fig. 26
The entity of an
Onebit_subtractor

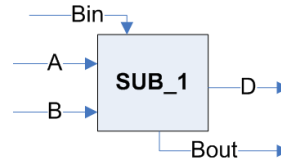
$$\begin{aligned}
 D &= A - B \\
 0 &\leq A, B \leq 2^n - 1 \\
 A &\geq B
 \end{aligned}$$



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.

Fig. 27
The entity of an
Onebit_subtractor

$$\begin{aligned}
 0 - 0 &\rightarrow 0 \\
 0 - 1 &\rightarrow 1, \text{ borrow } 1 \\
 1 - 0 &\rightarrow 1 \\
 1 - 1 &\rightarrow 0
 \end{aligned}$$



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

1 1 1 0 1 1 1 0	A (7..0)	(238)
- 1 0 1 1 1	- B (7..0)	- (23)
-----	-----	-----
= 1 1 0 1 0 1 1 1	D (7..0)	(215)

(starred columns are borrowed from)

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

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: **corrected**

- Use standard arithmetic blocks for integer numbers based on the 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 operands 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.

$$\text{a) } (+26)_{10} \quad + \quad (101010)_{2C}$$


$$\text{b) } (101010)_{2C} \quad - \quad (-21)_{10}$$

$$\text{c) } (+18)_{10} \quad + \quad (101110)_{2C}$$

$$\text{d) } (-31)_{10} \quad - \quad (010110)_{2C}$$

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.

Symbol	XC2C256 CoolRunner-II CPLD Parameter	-6		Units
		Min.	Max.	
T_{PD1}	Propagation delay single p-term	-	5.7	ns
T_{PD2}	Propagation delay OR array	-	6.0	ns

 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?

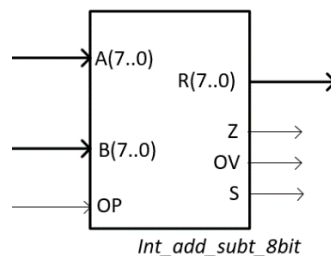


Fig. 28
The entity of an circuit to add and subtract 8-bit integer numbers.

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

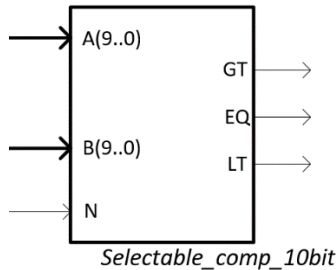
The idea is to design and implement a 10-bit comparator for both, radix-2 (natural numbers including the zero, also named whole numbers) and two's complement (2C) numbers (integers, which are positive or negative). The entity symbol is represented in Fig. 29 and shows the data select input N used to 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.

Fig. 29
The entity of a 10-bit comparator that works with natural or integer numbers.



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_1bit* 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.
5. Measurement of the maximum speed of processing.



4.4 Performing gate-level simulations and propagation time measurements

This project takes as example circuits for performing gate-level or timed simulations the *Adder_1bit* and the *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.

- Perform a gate level simulation in an *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.

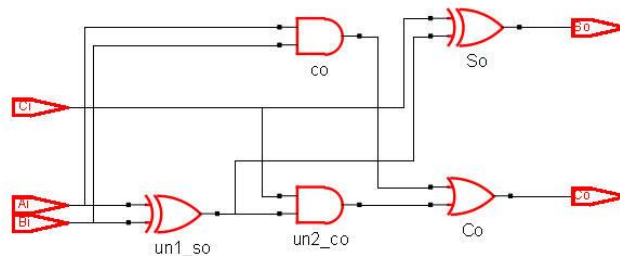


Fig. 30
Structural
Adder_1bit
consisting of several
levels of gates.

- Perform a gate level simulation in an *Int_add_subt_8bit* build using ripple carry techniques. Find the maximum speed of computation of this circuit for a given target PLD chip.
- Perform a gate level simulation in an *Int_add_subt_8bit* build using carry-lookahead techniques. Find the maximum speed of computation of this circuit for a given target PLD chip.
- Compare results for the same *Int_add_subt_8bit* when the target chip is a CPLD or a FPGA.

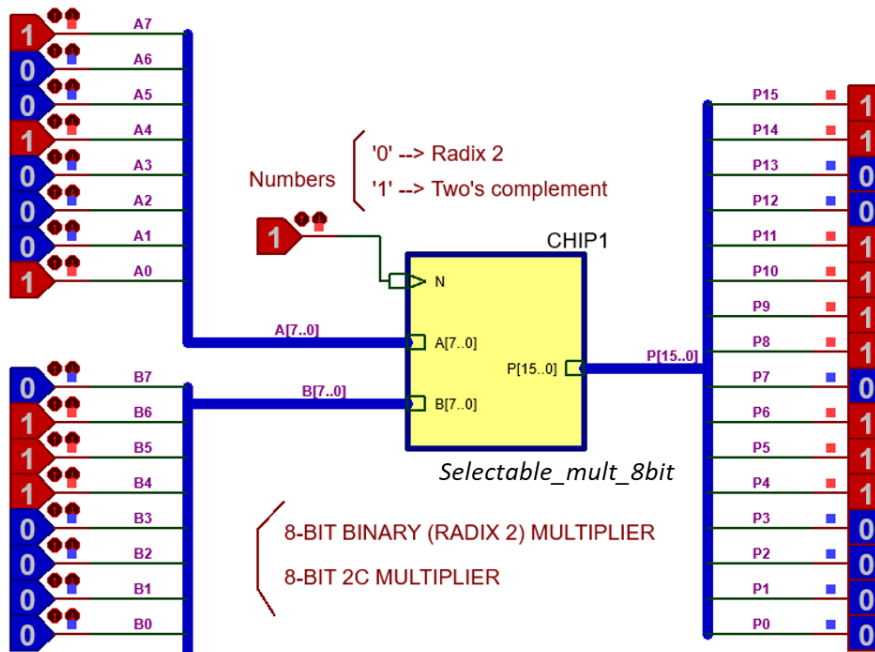


4.5 How to design an 8-bit multiplier for 2C integer numbers?

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.

Fig. 31
 Example of an 8-bit multiplier for 2C integers. The circuit has a control signal N to select which data operate: signed or unsigned integer. This is the link to the [circuit](#).



- 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](#).



→ 2



Sequential systems

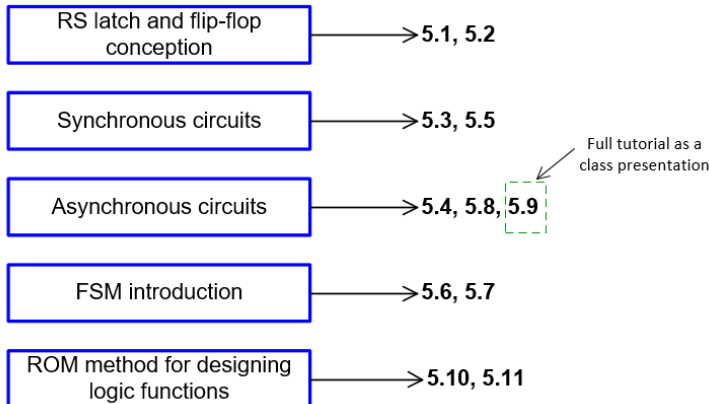


P5 1-bit memory cells: latches and flip-flops Objectives

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

- 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:



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.

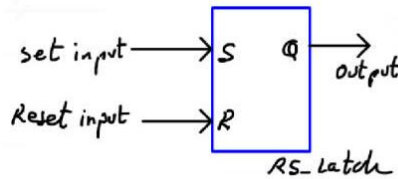


Fig. 32
Symbol and function
table of an RS latch.

R	S	Q ⁺ <i>↖ in the future (the next value)</i>
∅	∅	Q <i>the current value</i>
∅	1	1 <i>set to '1'</i>
1	∅	∅ <i>Reset to '0'</i>
1	1	Not permitted (undefined)

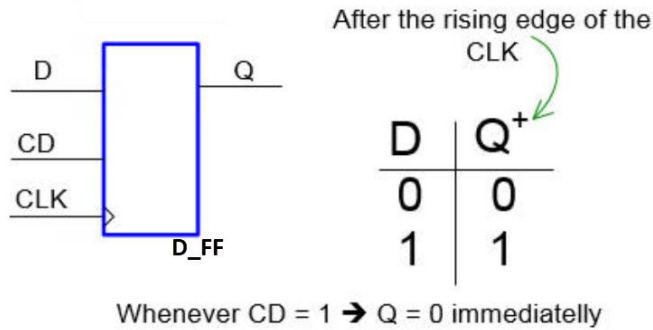


5.2 Data flip-flop (D_FF)

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?

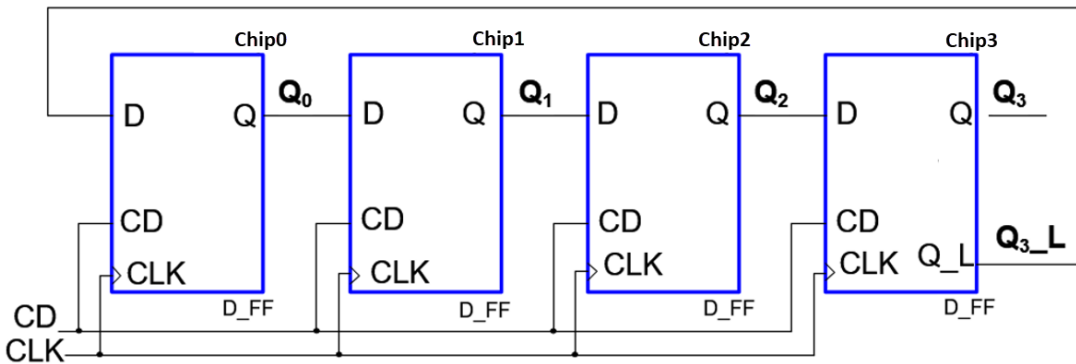


Fig. 34
Circuit based
on data flip-
flops (D_FF).

After the rising edge
of the CLK

D	Q ⁺
0	0
1	1

Whenever CD = 1
→ Q = 0 immediately

✚ Some insight into the solution of the problem can be found [here](#).

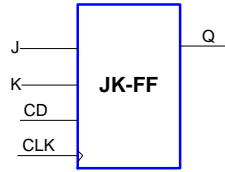


5.4 JK_FF and analysis of an asynchronous circuit

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.



JK	Q ⁺
00	Q
01	0
10	1
11	Q'

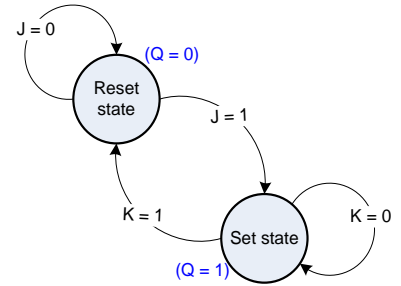
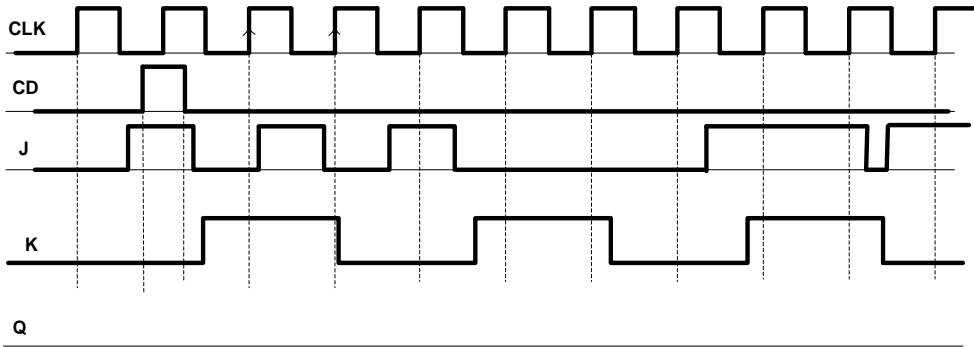


Fig. 36
Example input
waveforms.



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.

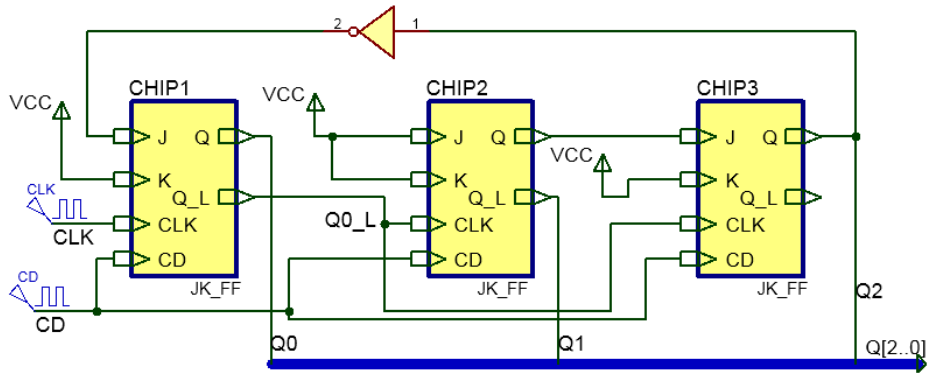
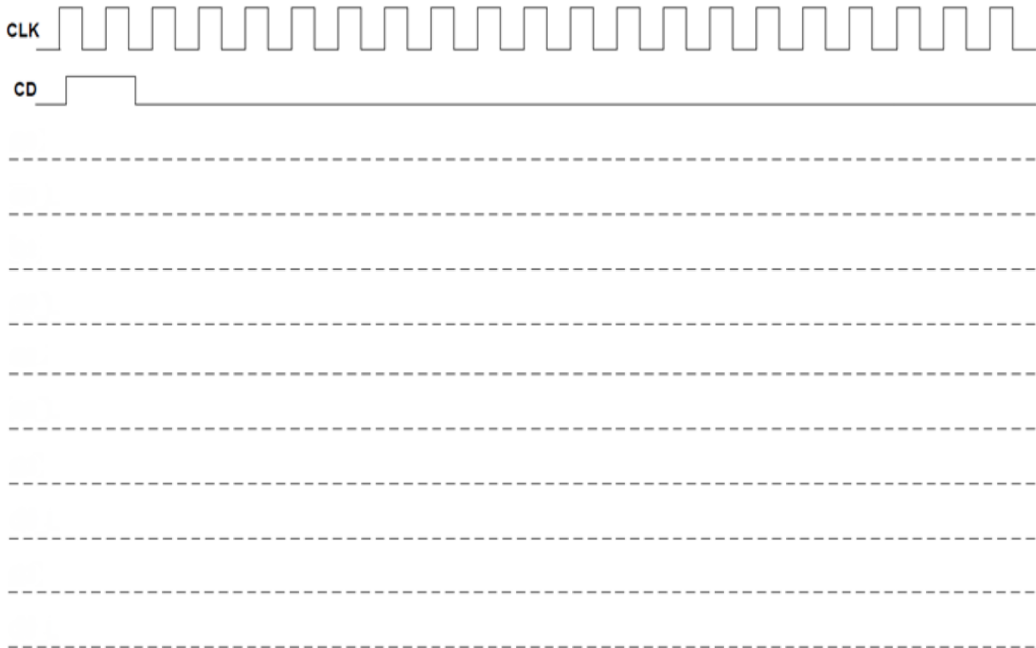


Fig. 37
Example of an asynchronous circuit.

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.

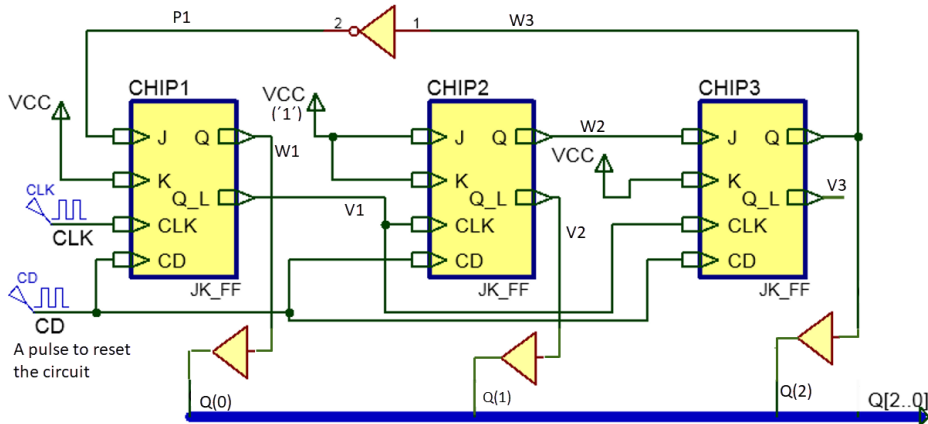


Circuit development in VHDL and testing

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.

Fig. 38
Example of the fully annotated circuit to be translated to VHDL. It can be called *async.vhd*



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.

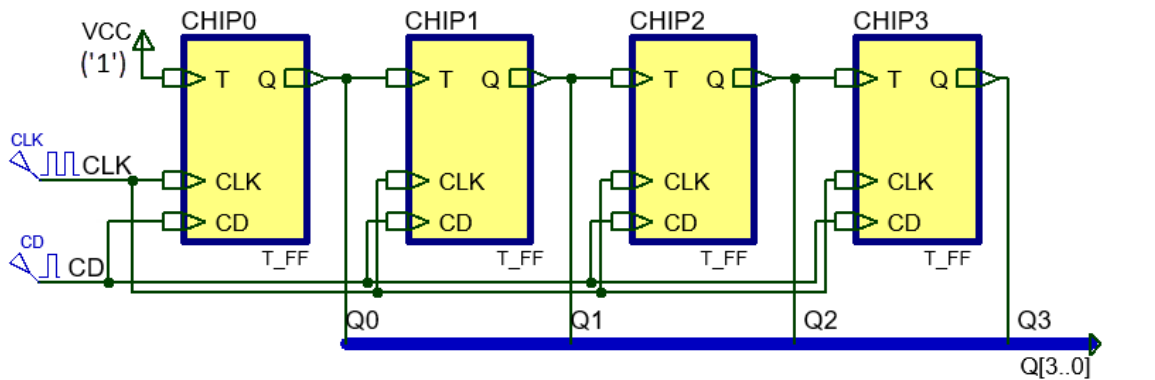


Fig. 39
Circuit based on
T_FF.



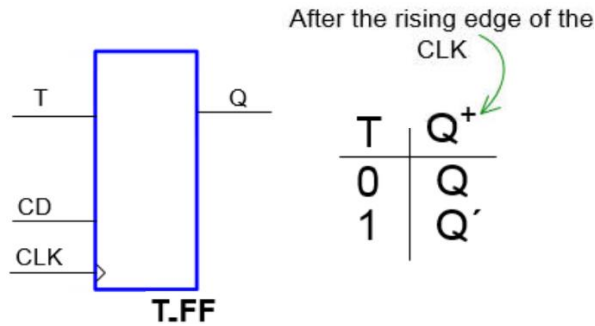


5.6 Design a toggle flip-flop (T_FF) using the FSM strategy

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.

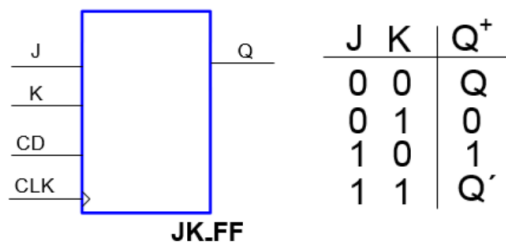


Fig. 41
Symbol and
function table of a
JK flip-flop
(JK_FF).



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.

Fig. 42
Symbol and function table of a synchronous toggle T-type flip-flop.

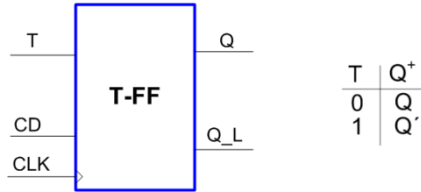


Fig. 43
This is the clocked circuit to be analysed. [Here](#) is a Proteus version of this circuit that can be simulated.

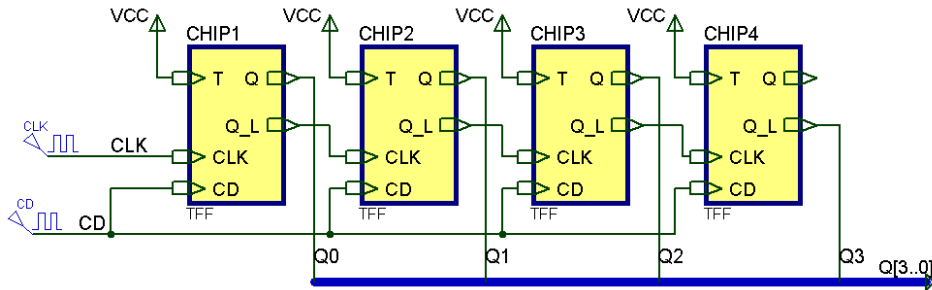
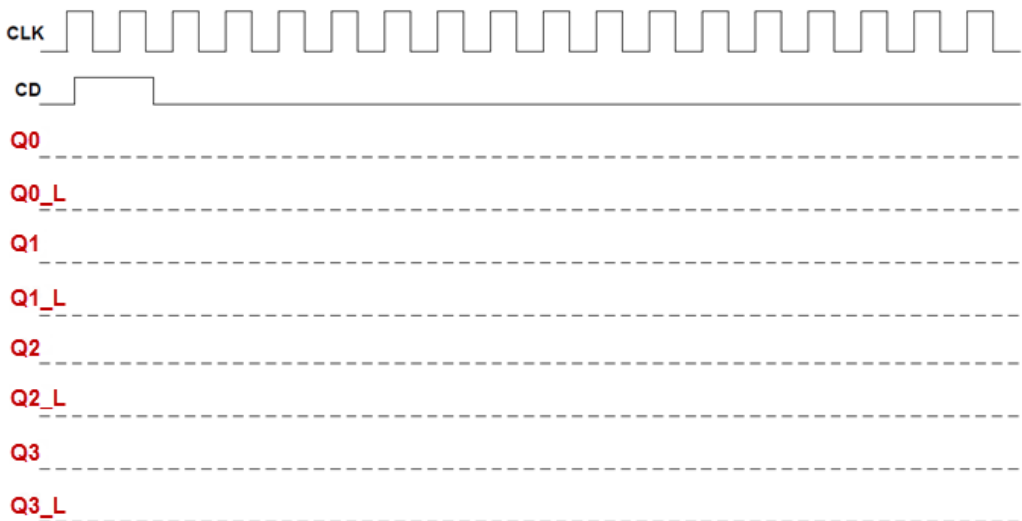


Fig. 44
Output waveforms to be deduced from the circuit in Fig. 43.



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).

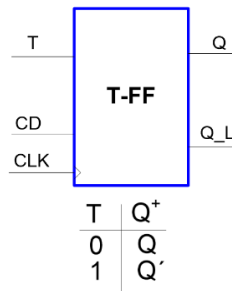


Fig. 45
Diagram and function table of a T_FF and example of an asynchronous circuit.

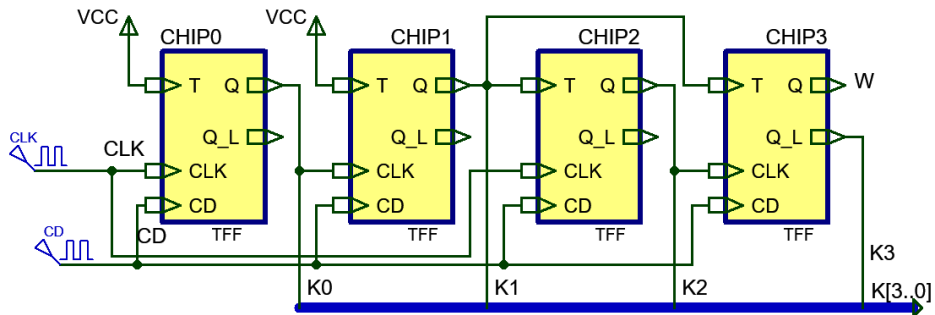


Fig. 46
Output waveforms to be deduced from the circuit in Fig. 45.

✚ 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: **corrected**

- Discuss the standard architecture of a finite state machine (FSM): current and next states, the state register, 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 simplify the interface for large keyboards.

+ A complete tutorial can be found [here](#).

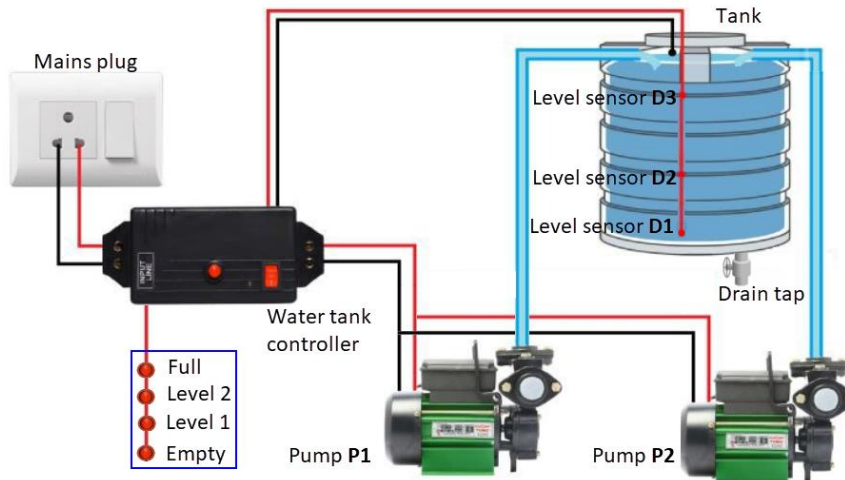


6.5 Water tank controller

- 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.

Fig. 47
Diagram of the
water tank
installation.

This link is an
example of a
[state diagram](#).



- 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 $\text{HEX0}(6..0)$ 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.

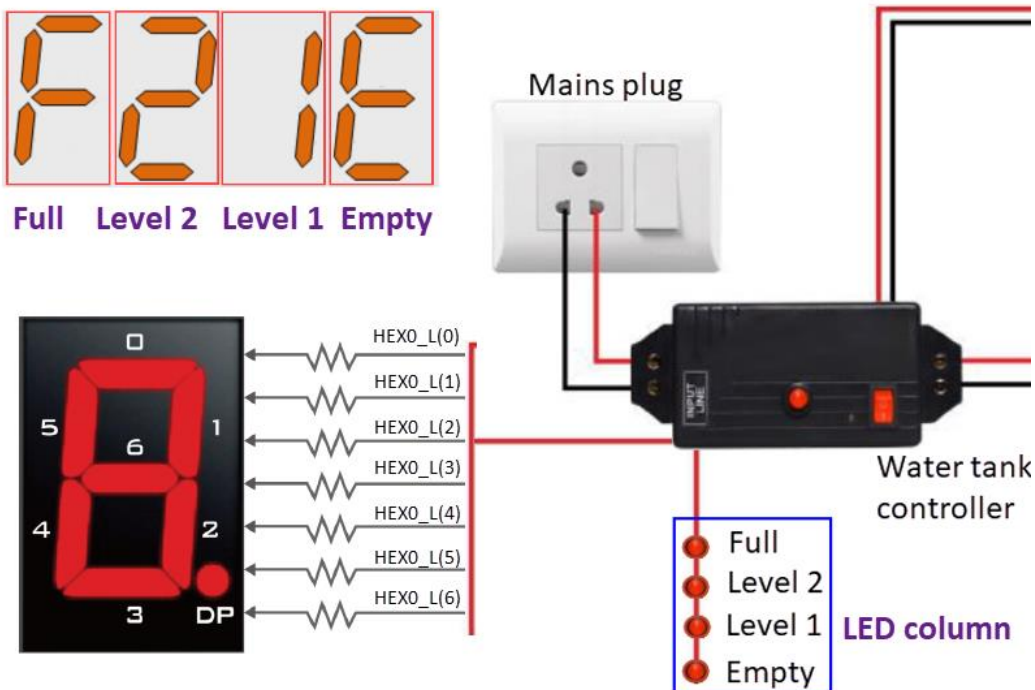


Fig. 48
Enhancement with a 7-
segment display.



6.6 Traffic light controller

This problem is discussed in the [tutorial](#).



6.7 Stepper motor controller

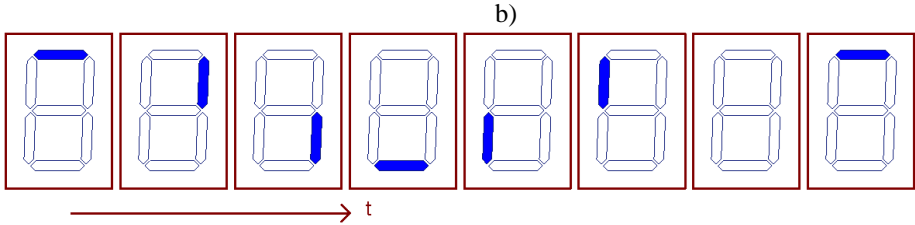
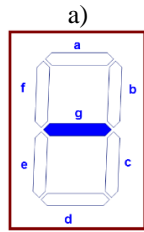
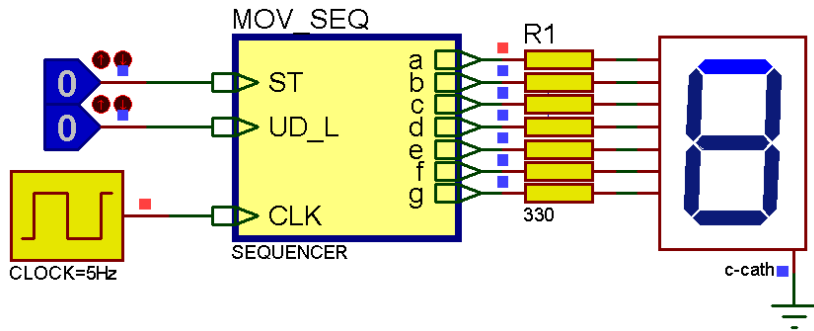
Reinvent project 0 using VHDL techniques as a canonical FSM.



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.

Fig. 49
a) Block diagram of the circuit.
b) Digit segments and position when idle. c) Sequence of switching LED segments for UD_L = '1' (up).



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: **corrected**

- Explain the meaning and use of signals such CE, LD, TC, Q, D_{in}, etc.
- Design synchronous canonical standard counters such as FSM.
- Design any kind of counter selecting one or several of the following strategies:
 - Plan X: as a typical enumerated FSM in P6.
 - Plan Y: as a scalable block based on STD_LOGIC_VECTOR signals.
 - Plan C2: using building blocks like standard counters and other components as in Chapter 1.
- 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.
-



7.1 1-digit BCD counter (flat)

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

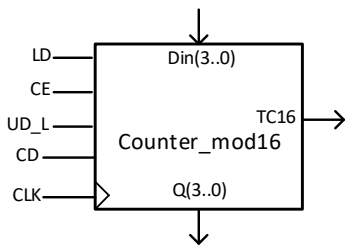
See the P7 project [tutorial](#).



7.2 Synchronous universal 4-bit binary counter

Our goal is to design as 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. 50. It is a chip similar to the classic 74LS169.

Fig. 50
Synchronous
4-bit universal
binary counter.
Symbol and
function table.
This is an
example
[Proteus](#)
[simulation](#) of a
very similar
circuit.



LD	CE	UD_L	Q*	Synchronous operation after the CLK's rising edge
1	x	x	Din	Parallel load (register data)
0	0	x	Q	Do nothing (inhibit)
0	1	1	$(Q+1)_{\text{mod}16}$	Up counting in binary
0	1	0	$(Q-1)_{\text{mod}16}$	Down counting in binary

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, state diagram.
How many states this sequential system must have?

Planning (plan Y)

1. Customise the general FSM architecture for this problem indicating where every input and output is connected. Plan the circuit as a FSM in a single VHDL file.
2. How many data flip-flops (D_FF) are required? Draw the schematic of the state register. Which is the internal encoding the *current_state* signal
3. Name the circuit *Counter_mod16* and plan the circuit as a FSM in a single VHDL file.
4. Write the truth table for the CC2 and propose an internal circuit.
5. Write the truth table for the CC1 and propose an internal circuit. Draw the truth table's flow chart as the behavioural interpretation.

Developing

6. Translate the circuit schematic to VHDL and start an EDA project to synthesise it for a given CPLD/FPGA target chip.
7. Inspect and analyse the RTL view and technology schematics. Check the number of DFF registers.

Test

8. Run a functional simulation translating the timing diagram into a VHDL test bench.
9. Run a gate-level simulation and determine propagation time from CLK to output (t_{CO}) and thus, the maximum frequency of operation for the given target chip.



+ Problem [discussion](#).

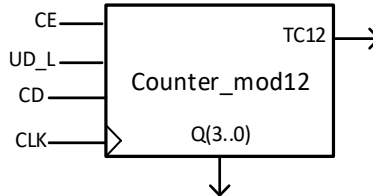
This [picture](#) is an example of a functional simulation of the universal counter modulo 16. The comments in red ink are very important to check whether the circuit works as expected.



7.3 Synchronous modulo 12 counter

Design as a complete project following the usual steps, the synchronous up/down modulo 12 counter represented in Fig. 51 using 2 different strategies. Compare and discuss the advantages and drawbacks of each strategy.

Fig. 51
Synchronous
up/down binary
counter modulo
12 with
asynchronous
clear direct.



Project X. Single VHDL file (flat) FSM based on naming states and the use of *State_type* enumerated signals. Thus, using this methodology this 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. Thus, using this methodology this project becomes another exercise like the *Counter_mod16* in 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 using the plan Y.

+ Here you are a [tutorial](#) on the design of a 4-bit data register.



7.5 Shift register

Design a synchronous 8-bit universal shift register using the plan Y.

 Here you are a [tutorial](#).

How three blocks *Shift_reg_8bit* can be connected to build a 24-bit shift register (plan C2)?



7.6 Hour counter for a real-time clock

Our goal is to design an hour counter to be used in a real-time clock device to count the hours in modes 0 – 12 ($M = '1'$) and 0 -24 ($M = '0'$). The Fig. 52 represents the schematic diagram of the application when connected to 7-segment digits.

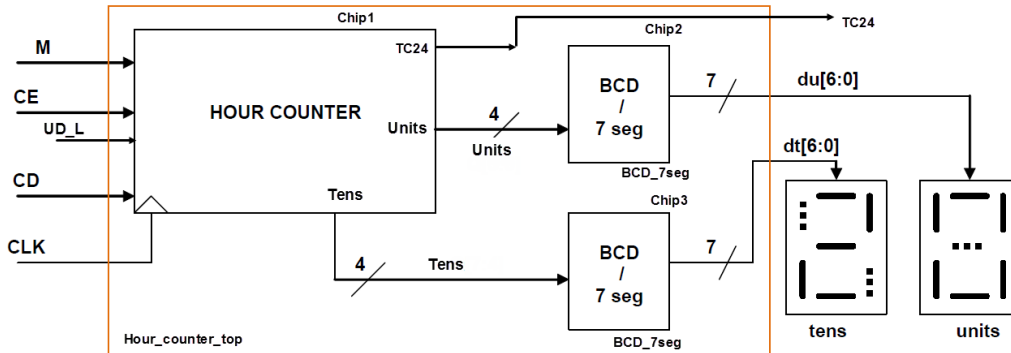


Fig. 52
Synchronous 2-digit
BCD module 12/24
counter

Specifications

1. Explain the function table of the *hour_counter* discussing the different modes of operation.
2. Draw an example of 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. Organize 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. 52?

Develop

6. Find the *Counter_mod16.vhd* file and translate the hierarchical schematic in Fig. 52 to VHDL.
7. Start a synthesis project and inspect the RTL and technology views schematics. Check the project summary to verify the number of DFF. The target chip may be any CPLD or FPGA available in the laboratory.

Test

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 the NEXYS 2 from Digilent. Assign pins and build and check the prototype



Hour_counter_top adding the necessary chips and modifications. Pay attention on how the 7-segment digits are wired.

✚ Problem [discussion](#).

This [picture](#) is an example of a functional simulation of the *Hour_counter* working in the AM-PM mode ($M = '1'$).

7.7 6-bit binary universal counter

Specifications

Our aim is to design the block represented in Fig. 53, a 6-bit synchronous binary counter (*Counter_mod64*) fully equipped with many features to make it versatile as a component in many projects.

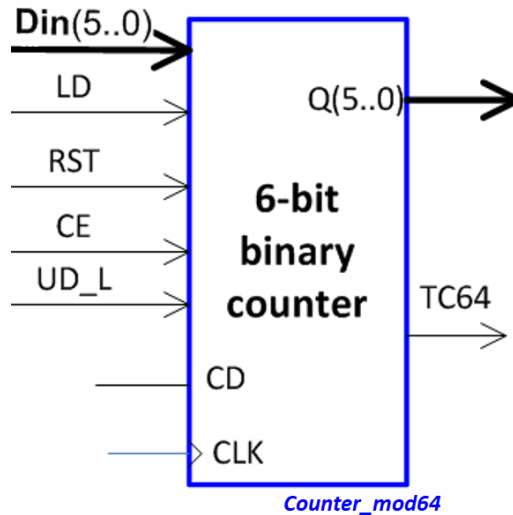


Fig. 53. The sequential block to be designed and its function table.

LD	RST	CE	UD_L	Q*	Synchronous operation
1	x	x	x	Din	Parallel load (register data)
0	1	x	x	0	Reset
0	0	0	x	Q	Do nothing (inhibit)
0	0	1	1	$(Q+1)_{\text{mod}64}$	Up counting in binary
0	0	1	0	$(Q-1)_{\text{mod}64}$	Down counting in binary

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 different modes of operation. How many states will this FSM contain?
2. Draw a timing diagram to represent the different modes of operation: Load, RST, count UP, count DOWN and do nothing.

Planning as a FSM in a single VHDL file (plan Y)

3. Draw the architecture of the FSM particularised for this problem and indicate where all the inputs and outputs are connected.
4. How many registers (D-type *flip-flops*) will the system include? Explain your answer.
5. Draw the truth table and flow chart of the combinational circuit CC1
6. Draw the truth table and flow chart of the combinational circuit CC2.



Development using CAD/EDA tools

7. What is the synthesis process? In which way can we examine the result of the synthesis process?

Verification:

8. Write the main features of a test bench file "Counter_mod64_tb.vhd" necessary to perform a functional simulation. For instance, translating to VHDL some vectors from the 2 section.
9. What are the "VHO" and the "SDF" files? Which is the use of these files? The target chip is the ispMACH4128V which has D_FF 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 if operation? Explain your answer.



7.8 Johnson counter

Design a synchronous 5-bit Johnson counter with count enable and reversibility control signals as shown in the symbol in Fig. 54 using both the Plan X (FSM strategy enumerating/labelling states) and the Plan C2 (using a standard component like the [Counter_mod16](#) in problem 7.2).

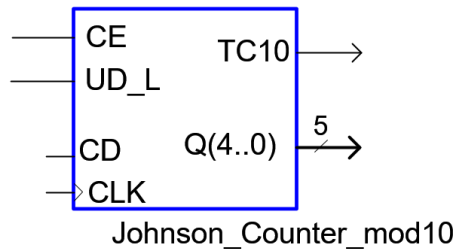


Fig. 54
Symbol of the
sequential system

- a) Draw the Johnson code for 5 bits. Draw the state diagram indicating both, transitions and outputs. Draw the function table for this counter. Draw an example of a timing diagram.

Plan X:

- b) Draw the FSM structure consisting of CC1 and CC2 and the state register. Indicate its inputs and outputs. How many D_FF will contain the state register if the internal states are coded in binary, and if they are coded in one-hot?
- c) Draw the CC1 truth table and its equivalent behavioural representation in a flow chart.
- d) Write the main VHDL sentences of CC2.

Plan C2:

- e) Propose an internal architecture based on the *Counter_mod16* and other blocks combinational blocks. A good idea is to design firstly the counter with only up counting direction. And secondly, complete de counter adding the reversibility feature.
- f) Annotate completely the schematic, chips, signals, etc. and leave it ready for VHDL translation. How many VHDL files will this project include? How many D_FF will this counter include?

For both design plans:

- g) Write the VHDL code 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 comments. Check the number of D_FF used.



- h) Use a VHDL test bench and run an EDA functional simulation to check how the circuit behaves in time. Print the timing diagram and make annotations.
- i) Which is the maximum frequency that can be assigned to the CLK signal when performing a functional simulation?
- j) Which is the maximum frequency that can be assigned to the CLK signal when performing a gate-level simulation if the target chip is an Altera CPLD Max II EPM2210F324C3 that has the following characteristics:

<i>MAX II Device Features</i>	<i>EPM2210</i>
$t_{PD\ gate}$ (ns)	1.7
t_{CO} (ns)	4.6

+ Problem [discussion](#) using plan X.

7.9 PIC18F4520 TMR2 prescaler design

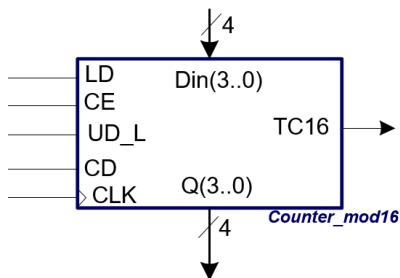
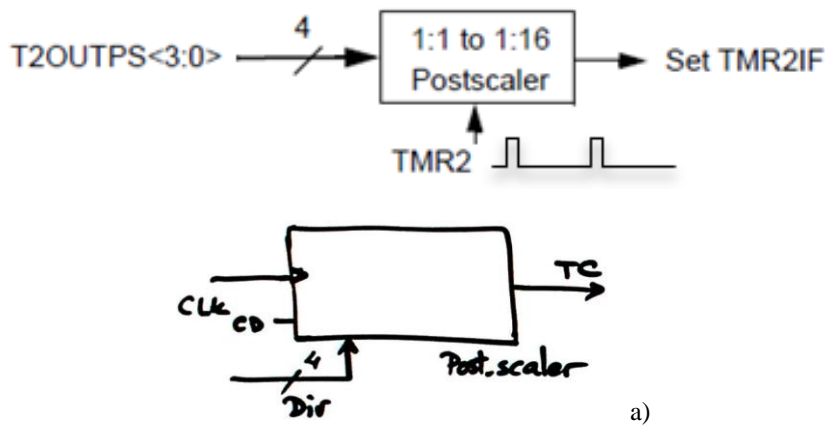
Design (specify and plan) the programmable *Post_scaler* available in a PIC18F TMR2 represented in Fig. 55 using VHDL techniques and structural plan C2 for a target FPGA chip. Use standard sequential and combinational circuits and logic gates. Let us rename the symbol and ports: TMR2 is the **CLK** input to the circuit, T2OUTPS(3..0) is the frequency division selector **Div(3..0)**, and Set TMR2IF output is the terminal count **TC**.

The *Post_scaler* can divide from 1:1 to 1:16 depending on Div(3..0) binary value from “0000” to “1111”. For instance, if FCLK = 15 kHz and Div(3..0) = “0100”, the circuit becomes a 1:5 frequency divider generating FTC = 3 kHz; if FCLK = 84 MHz and Div(3..0) = “1011”, the circuit becomes a 1:12 frequency divider generating FTC = 7 MHz.

Fig. 55

a) TMR2 postscaler block schematic from Microchip and our adapted symbol.

b) Symbol and function table of component Counter_mod16 that may be used in this structural design.



LD	CE	UD_L	Q ⁺	
1	x	x	Din	parallel load
0	0	x	Q	do nothing
0	1	1	$(Q+1)_2$	up (mod16)
0	1	0	$(Q-1)_2$	down (mod16)

b)

+ Problem [discussion](#) using plan C2 where up to three designs are proposed.





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 an FSM.
- Design complex circuits or dedicated processors based on the 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 circuits that can be designed using the techniques and strategies described in this P8 section.
- Explain why a microprocessor is a programmable dedicated processor.

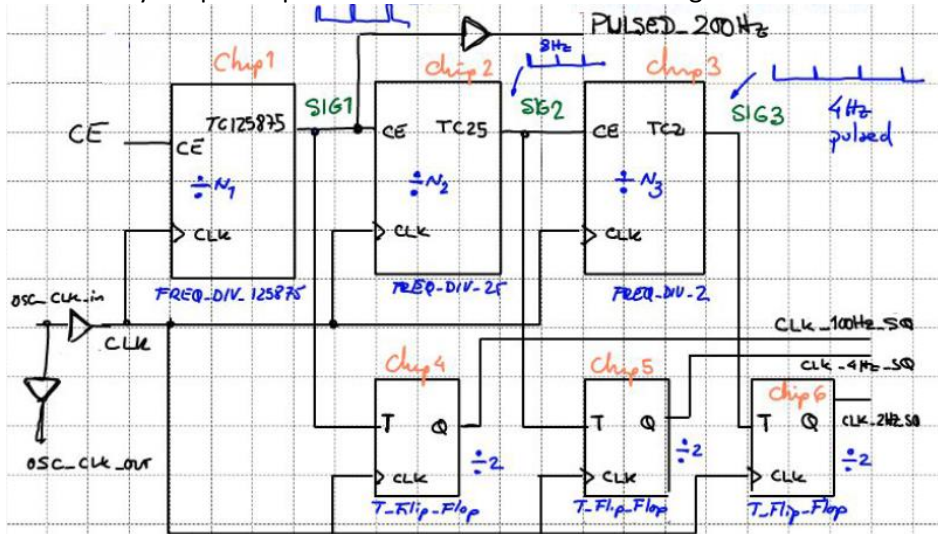


8.1 Generation of CLK signals

The sketch in Fig. 56 represents the internal architecture of the *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?

Fig. 56
Proposed internal architecture for a generator of CLK signals.



- 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. 56 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. 57 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. 58 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.

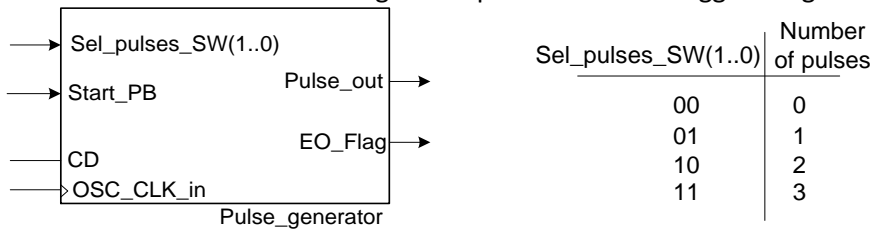


Fig. 57
Symbol and function table for the proposed pulse generator.

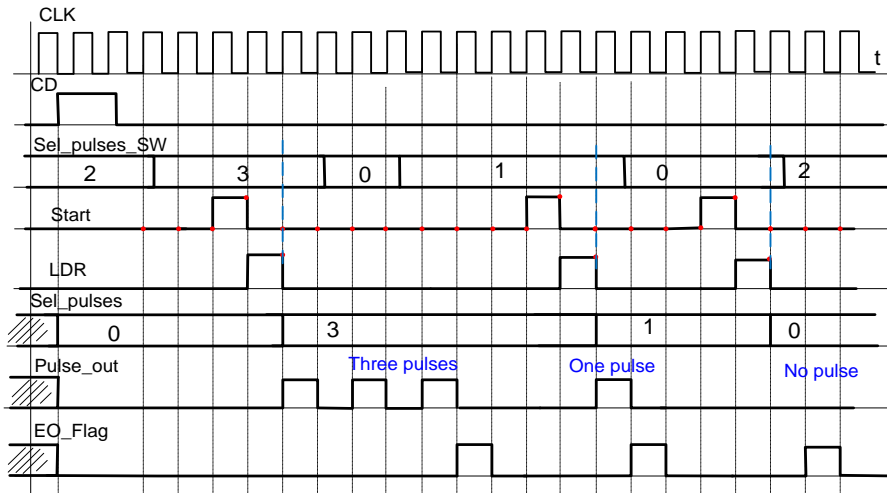


Fig. 58
Example of a timing diagram showing circuit's activity. Start is a synchronous single pulse derived from the output of a debouncing filter.

The architecture in Fig. 59 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?

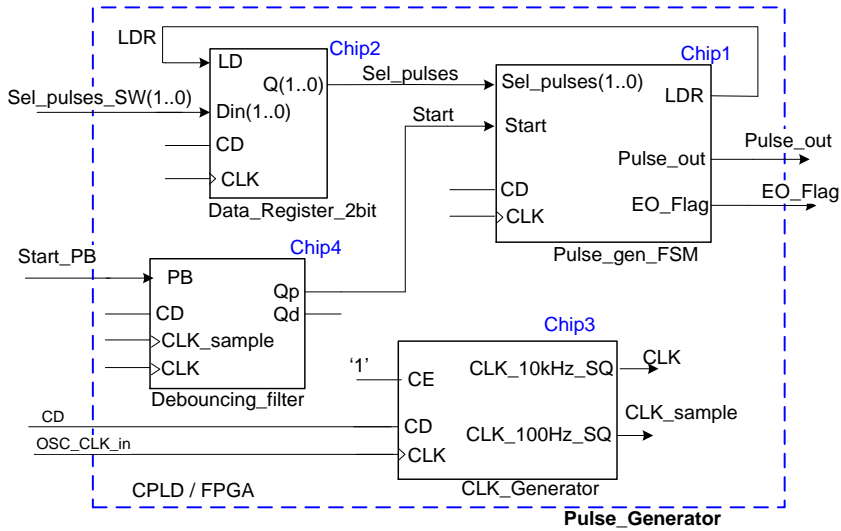
Explain the internal architecture of the *Data_register_2bit* component. How many states does it have? How many data flip-flops (DFF) are required to implement its state register?

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.



Fig. 59
Sketch of an
internal
architecture
for the
system.



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. 58 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. 62 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.

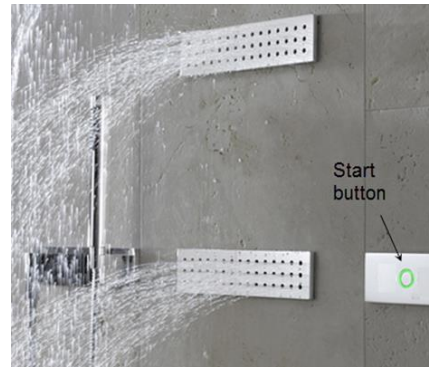
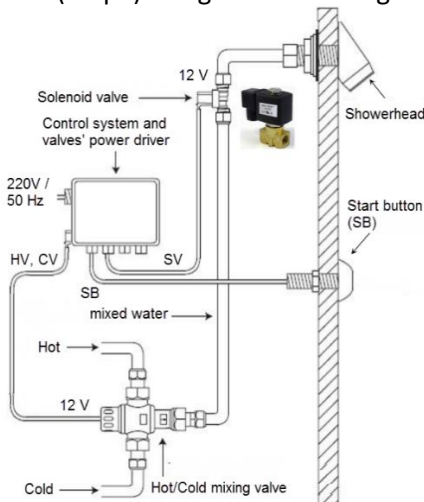
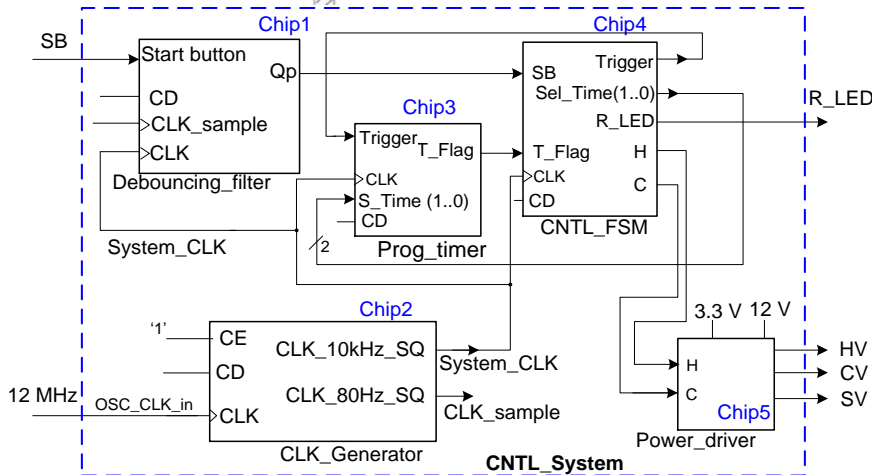


Fig. 60 Photograph and sketch of the shower installation to be automated.





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_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. 61 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.

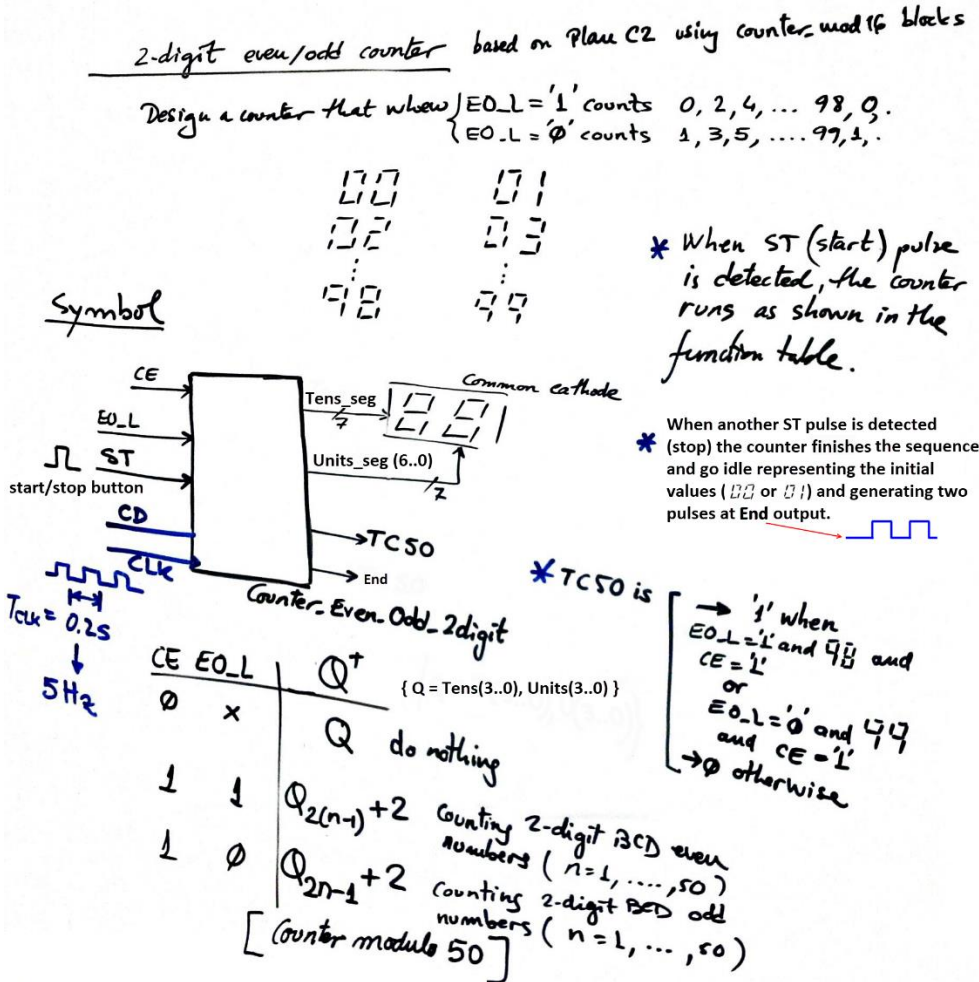


Fig. 61 Symbol of the proposed counter and function table when the counter is running.

You can draw several timing diagrams to get a better idea on how the system works, and also a state diagram for the *Control_unit* block is required to figure out how the status and control signals work.

These is a feasible [plan](#) that relies on previous CSD components and projects.

You have two options to design the binary *Counter_mod50*:

- Using the plan Y in a single VHDL file.
- 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.](#))



→ 3



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 combinational circuits 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 diagrams and flowcharts to C language for writing 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. 62.

- 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.

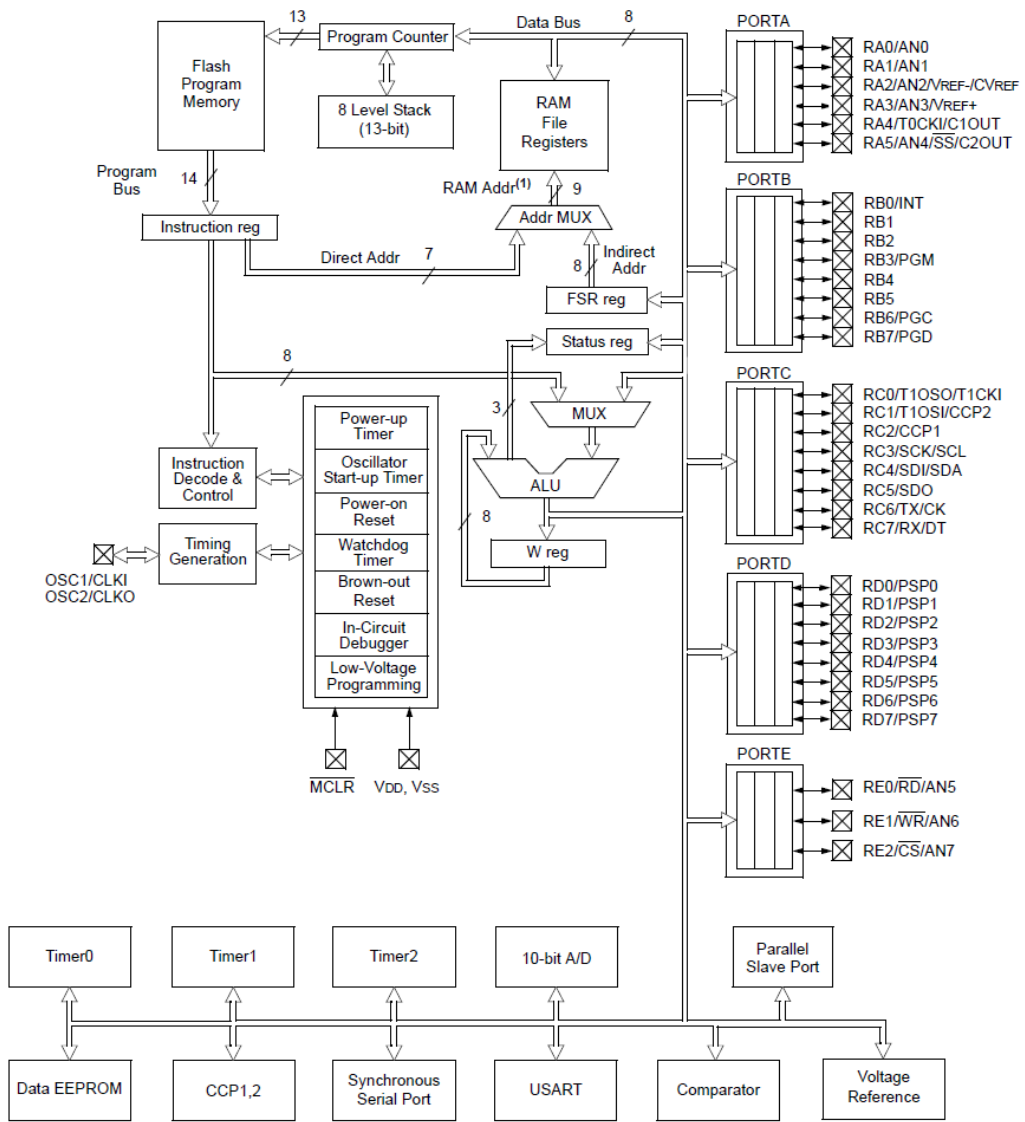


Fig. 62
PIC16F87xA
architecture.

Device	Program Flash	Data Memory	Data EEPROM
PIC16F874A	4K words	192 Bytes	128 Bytes
PIC16F877A	8K words	368 Bytes	256 Bytes



Fig. 63
PIC16F
Assembler RISC
instruction set.

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



9.2 Invent a Dual_MUX4 based on a μC

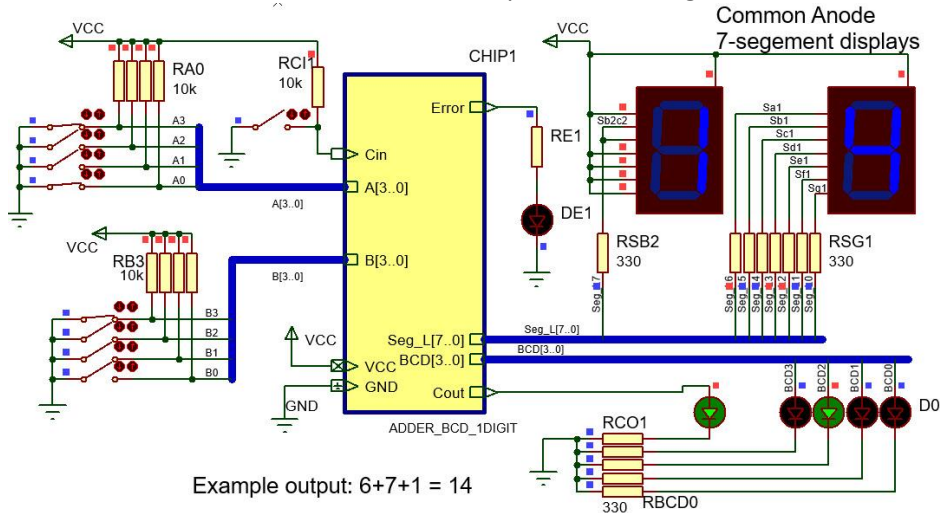
This assignment and tutorial can be found [here](#).



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. 64.

Fig. 64
1-digit BCD adder.



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 → 5p.
- Development and test → 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 show 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.

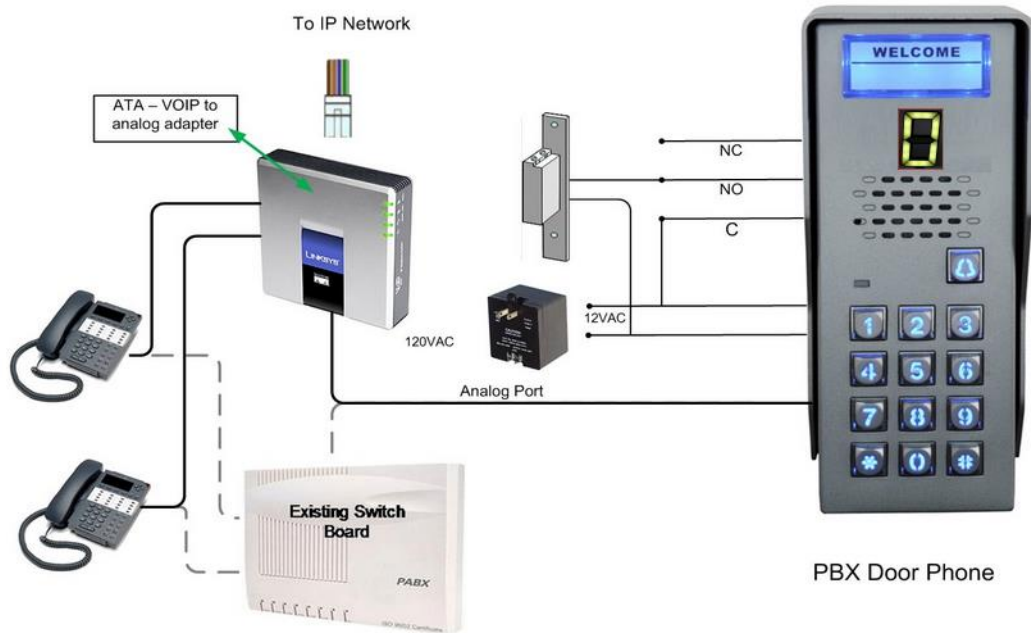
And on top of that, be fair: you are here at this university to learn the content of this subject deeply, not superficially. A hardware engineer has to be able to project circuits professionally and successfully, and to meet this goal, it is required a complete personal involvement and engagement. You 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. 65. 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.

Fig. 65
The keyboard that that has to be interfaced to a commercial PBX and the commercial [reference](#) of the diagram..



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. 66.

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.

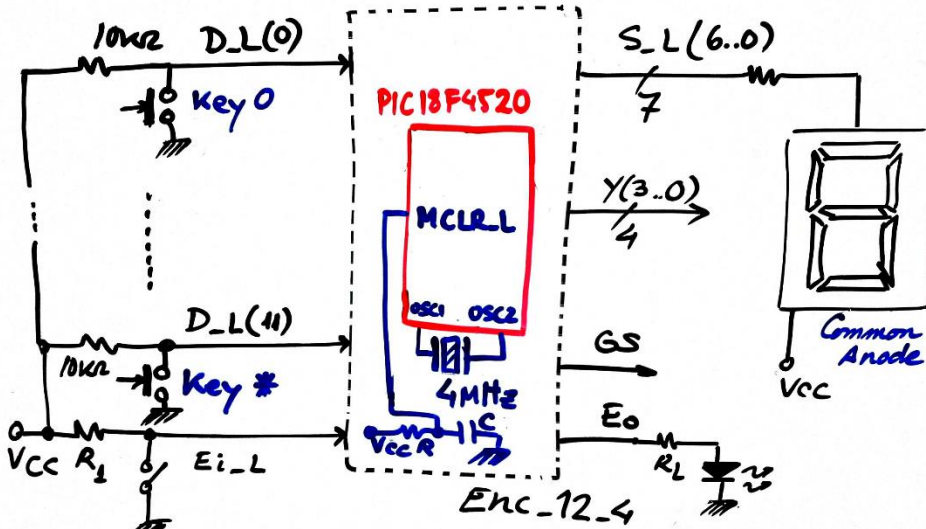


Fig. 66
Symbol of the *Enc_12_4* and the idea of designing it using a μC .

You can solve in a phase#1 the circuit with GS, Ei, Eo, and the BCD outputs Y; and later in a phase#2 you can add the feature of the 7-segment interface including the other S_L signals.

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 an FSM can be organised and solved using a microcontroller.
- Detect events like rising or falling edges using interrupts.
- Explain the interrupt logic of a microcontroller and the function of the interrupt service routine *ISR()*.
- Draw the hardware-software diagram of an application based on FSM.
- Redesign applications based on sequential systems using microcontrollers and C language.



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. 67. 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. 68). $Serial_out$ is held high when idle.

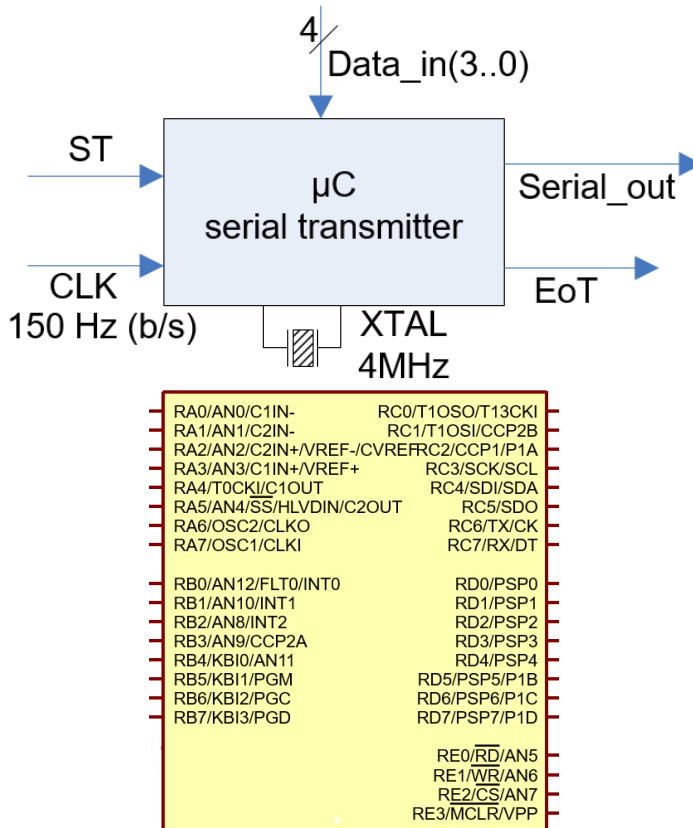
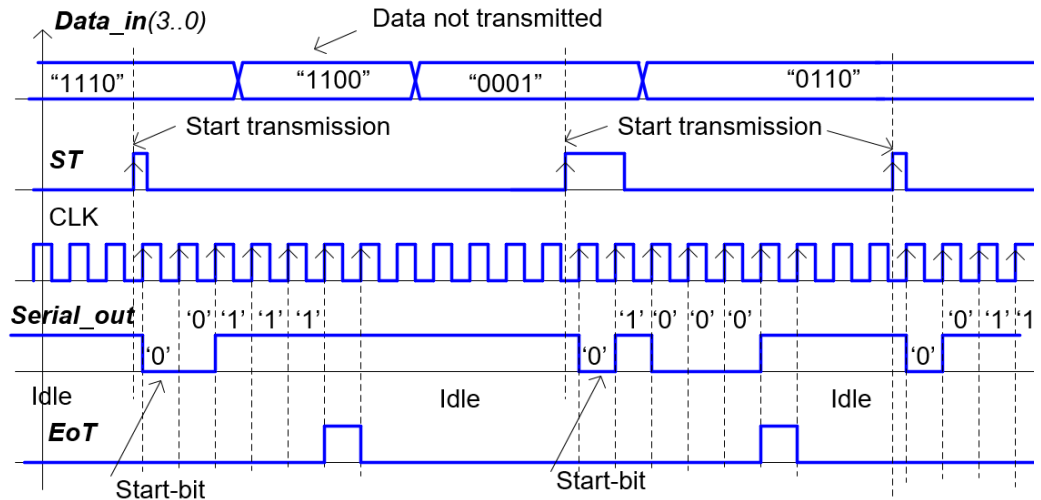


Fig. 67 Symbol of the data transmitter and the μC PIC18F4520 from Microchip.

- 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. 68
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.



- 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 *INT0IF* so that a new bit is transmitted at a time at the *Serial_out* as represented in Fig. 68. Transmission speed is 150 b/s.
- How *read_input()* works to generate the *char* variable *var_Data_in*?
- How the variables *var_Serial_out* and *var_EoT* are written to the corresponding pins using *write_outputs()* without interfering the other μ C port pins?
- Which is the *ISR()* used for? Propose the flow chart.
- 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.
- Draw the truth tables and their equivalent flow charts for the *state_logic()* and *output_logic()*.
- 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

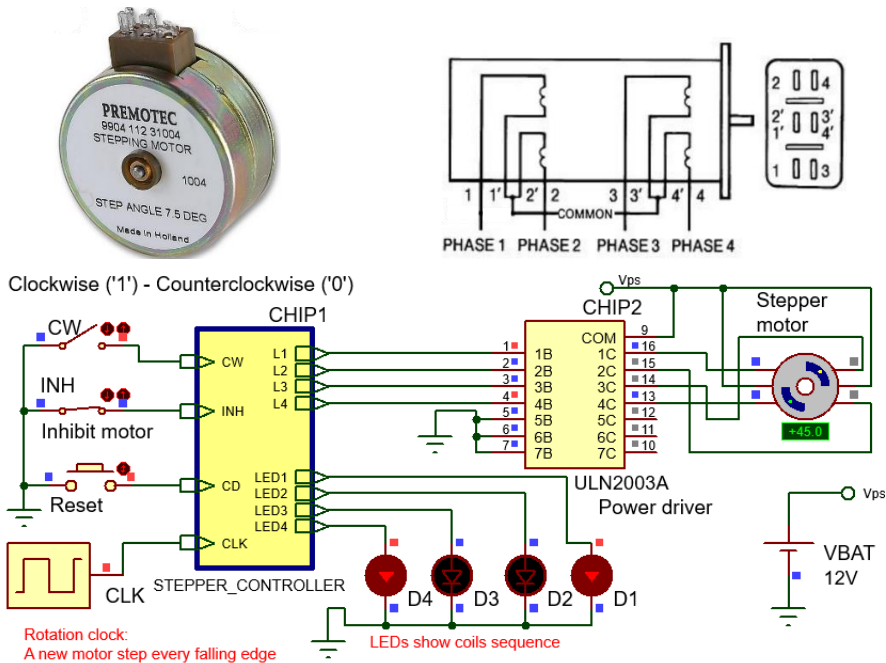
✚ A tutorial on this project is available in these references ([1](#)), ([2](#)).



10.5 Stepper motor controller

Design the digital control unit (*stepper_controller*) for the “9904 112 31004” stepping motor from Premotec shown in Fig. 69 following our microcontroller-based strategy. Today [stepper motors](#) can be found in computer peripherals, machine tools, medical equipment, automotive devices, or small business machines, to name a few applications. Clockwise (**CW**) and counter-clockwise (CCW) rotation can be achieved by reversing the step sequence. Inhibit (**INH**) is like a count disable, do not letting the motor rotate. Step or stride angle is 7.5 degree, thus 48 **CLK** periods are required for a full revolution. External **CLK** frequency is 96 Hz, and so when running it rotates at 2 revolutions per second (Fig. 73). The idea is to connect four port pins to the motor coils and drive them with the right sequence so that the motor inhibits or rotates clockwise or counter-clockwise accordingly to the input signals **INH** and **CW**. Four additional pins are used connected to LED to visualise the coils binary sequence.

Fig. 69 Example of two-phase stepper motor: characteristics, connections and unipolar winding circuit using a power driver to energise coils.





	L4	L3	L2	L1
1	ON	OFF	OFF	ON
2	OFF	ON	OFF	ON
3	OFF	ON	ON	OFF
4	ON	OFF	ON	OFF
1	ON	OFF	OFF	ON

FULL STEP
 ↓ ↑
 CW CCW

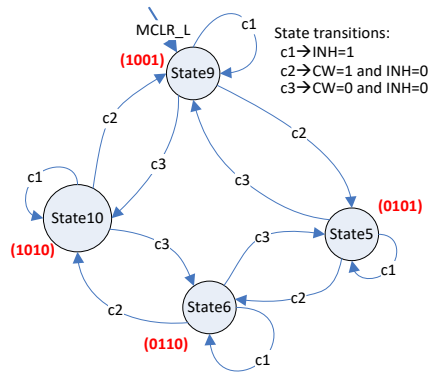


Fig. 70 Full wave stepping sequence and unipolar winding circuit using a power driver to energise coils. Proposed state diagram to control the FSM.

1. Draw the schematic: input switches, outputs, reset (MCLR_L) and 4.8 MHz quartz crystal oscillator OSC.
2. Draw the hardware-software diagram. Why the rotation CLK block has to be connected to RBO/INT pin? What the interrupt service routine *ISR()* is used for?
3. Organise and name RAM variables for the project. Explain how to configure port pins and interrupts in *init_system()*.
4. Explain how to poll the input values using bitwise operations in *read_inputs()*.
5. Explain how to drive the eight outputs using bitwise operations in *write_outputs()*.
6. Draw the truth table and flowchart for the *output_logic()*.
7. Draw the truth table and flowchart for the *state_logic()*.
8. Replace the external CLK configuring the embedded TMR0 (Fig. 71 in 8-bit mode) to obtain the same 96 Hz step frequency.

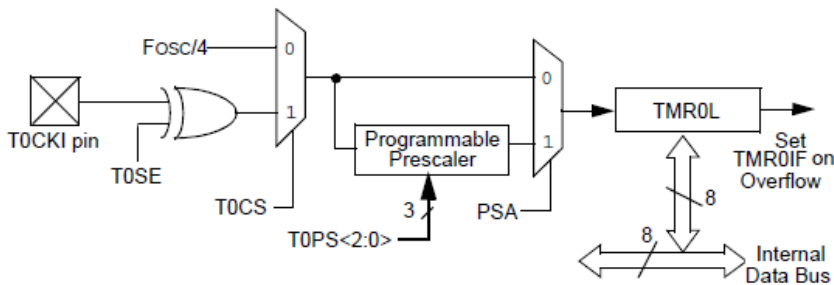


Fig. 71 TMR0 schematic in 8-bit mode.

+ Problem [discussion](#).





P11 Peripherals: LCD display

Objectives

After studying the content of this chapter on LCD peripheral, you will be able to:

- Explain the basics of an LCD display and its speed of operation.
- Design the electrical LCD interface to a microcontroller.
- Compile multiple-file projects integrating LCD peripheral C libraries.
- Enhance projects from previous units with an LCD for printing ASCII static messages.
- Use a variable (*var_LCD_flag*) to print only when there is new information.
- Use `<stdio.h>` functions to format strings of characters.
- Print dynamic data from several types (*int*, *double int*, *float*) on the screen.
- Connect an LCD to a microcontroller (Arduino, ATmega328P) using a 2-wire I2C interface.



11.1 LCD display using ASCII messages and static data

+ A tutorial on this project is available in this reference ([1](#)).

11.2 LCD display using dynamic data

+ A tutorial on this project is available in this reference ([1](#))

11.3 Interfacing an I2C display

+ A tutorial on this project is available in this reference ([1](#))



P12 Peripherals and complex applications

Objectives

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

- Explain the TMR0 architecture and how to configure it for timing and counting applications.
- Deduce the timing period (T_P) design equation and how to generate an arbitrary large counting modulo using RAM memory post-scalers.
- Determine for the TMR0 the systematic software overhead when timing.
- Explain the TMR2 architecture and how to configure it for precision timing applications.
- Explain how to adapt applications based on dedicated processor architectures (P8) to microcontrollers.



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. 72. 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.

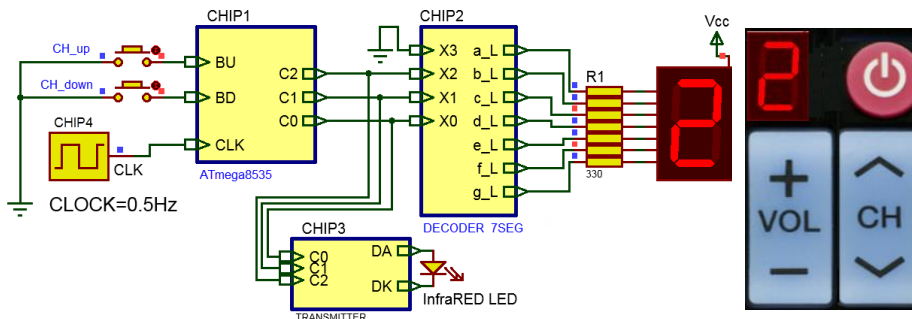


Fig. 72
Schematic of a basic remote control for 7 channels.

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.
4. Architecture of the software. Program variables. Use of interrupts.
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. 73. 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.

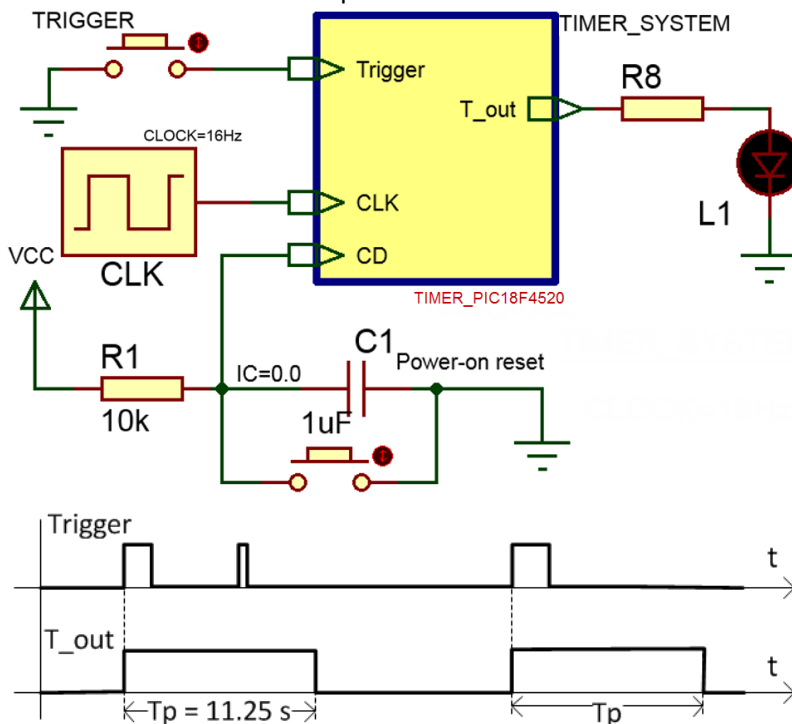


Fig. 73 Circuit symbol and waveforms for the "Timer" project. In option B the CLK (INT1) is not connected because the internal Timer0 is used instead.

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. 74 shows the symbol of an application based on the PIC18F4520 (Fig. 75) 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. 76 shows an idea of the state diagram.

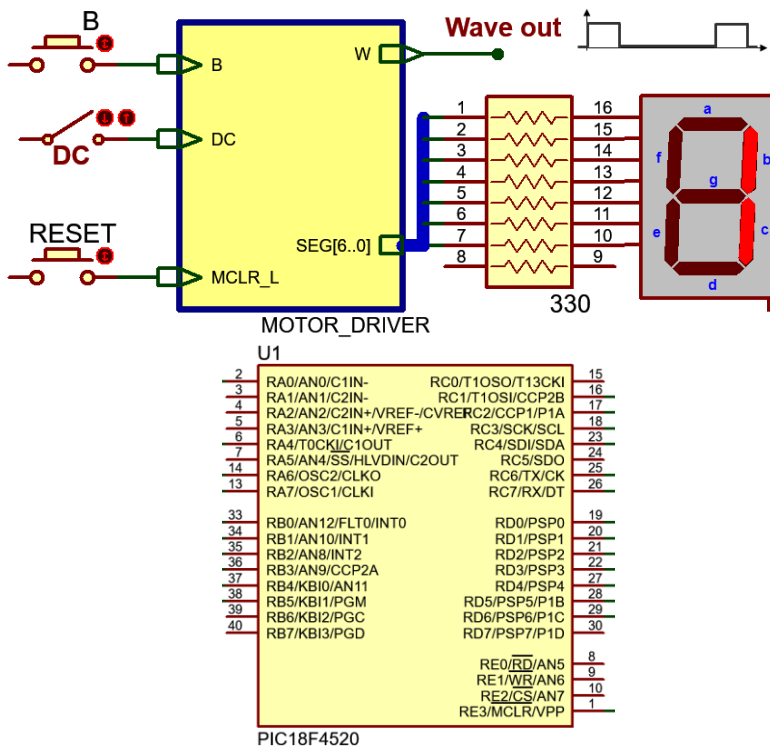


Fig. 74
This is an idea of the state diagram proposed to run this application. It must be completed.

Fig. 75
Pinning of the microcontroller PIC18F4520

- Draw the two waveforms indicating the T_{ON1} , T_{OFF1} , T_{ON2} and T_{OFF2} periods of time.
- 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()*.
- 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?

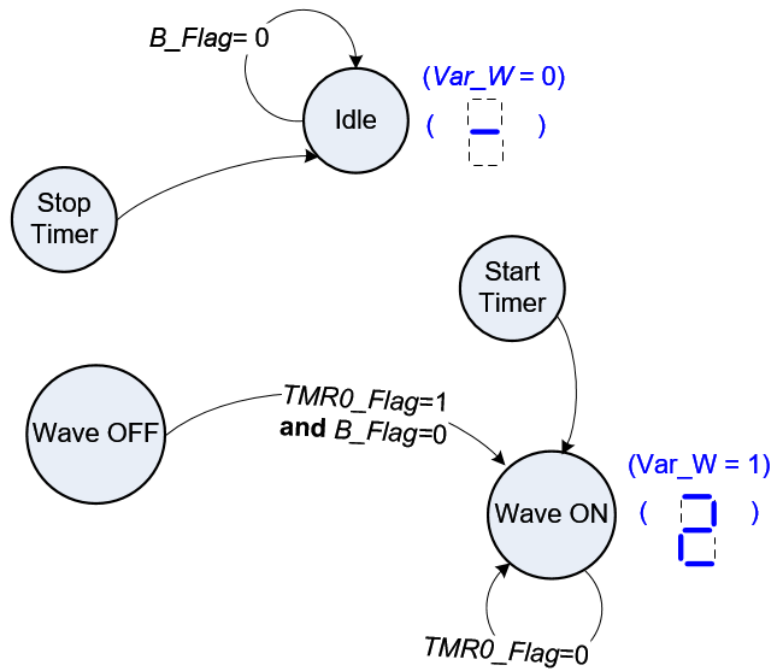
$$Timing_period = (4/F_{osc}) \cdot N1 \cdot N2$$
- Draw the hardware/software diagram indicating the required RAM variables and how the FSM is solved in software. How to implement



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. 76 and deduce the truth tables for the main functions of the C code: *state_logic()* and *output_logic()*.

Fig. 76
This is an idea of the state diagram proposed to run this application. It must be completed.



+ Problem [discussion](#).



12.5 Temperature measurement using timers

12.6 Temperature measurement using A/D converters



→ 4



Bibliography and internet links

Bibliography

- [1] Hwang, E. O., Digital logic and microprocessor design with VHDL, Thomson, Toronto, CA. 2005.
- [2] Tinder, R. F., Engineering Digital Design, 2nd ed., Academic Press, London, UK, 2000.
- [3] Reese, R. B., Microprocessors, from Assembly language to C using the PIC18Fxx2, Da Vinci Engineering Press, Hingham, USA, 2005.
- [4]
- [5]
- [6]

Internet links

- [7] CSD [learning objectives](#).
- [8] Indications for [professional communications](#) by email.
- [9] [Thunderbird](#) e-mail client.
- [10] [Firefox](#) internet browser.
- [11] [Google Suite](#) cloud file system and applications.
- [12] CSD [software](#).
- [13]



This list has to contain all the CSD topics

Behavioural design approach, 21

Encoders, 21

 group select (GS), 21, 28

priority high, 21

Flat design approach, 21

Product-of-sums (PoS), 21

Standard combinational circuits

 74HC147 - 10 to 4 line priority
 encoder, 27

Sum-of-products (SoP), 21

Truth table, 21