C.L.I.L. MODULE

C PROGRAMMING APPLICATIONS TO SYSTEMS ANALYSIS

Abstract: this CLIL module shows some applications of C programming in the fields of linear systems analysis and of software simulation of simple finite state machines. Two main applications will be analyzed in detail: the step response of first and second order linear systems; the simulation of the behavior of simple finite state machines. Each Section of the module presents an application and is concluded by some exercises and activities on the same topic.

Glossary:

Array: a set of ordered variables of the same type indexed with an integer number.

FILE pointer: memory address reserved by the program for the data contained in the file.

AC, DC and transient analysis: three basic kinds of electronic circuits analysis that can be performed with a CAD tool. DC analysis is useful for circuit transfer curve calculations (for instance DC output voltage vs. DC input voltage characteristics). AC analysis is useful for circuit frequency response calculation (output voltage amplitude and phase vs. frequency – the input is sinusoidal with a fixed amplitude). Transient analysis is useful for circuit transient response calculation (output voltage vs. time – the input is a time varying voltage).

Pulse, step and ramp transient responses: output voltage waveforms vs. time of an electronic circuit when its input is a Dirac pulse, step or ramp waveform, respectively.

Transfer function: it is the ratio between the output and input signal Laplace transforms of a Linear Time Invariant (LTI) system. It provides useful information about the stability of the system.

Zeros and poles: The zeros of the rational transfer function of an LTI system are the roots of its numerator. The poles are the roots if its denominator. The stability of the system depends on its poles and zeros values.

Audio amplifier: it is a power amplifier used to amplify low power audio signals to a power level suitable for driving loudspeakers.

Relay: it is a high power switch that can be opened/closed by means of a low-power electronic signal.
**Drive capability:** it is the maximum output current a circuit is able to source/sink. In digital circuits it limits the number of logic gates that can be connected to the circuit output (fan-out), while in analog circuits it provides a lower limit for the equivalent resistance of the load of the circuit.

**Non-volatile memory:** it is an IC semiconductor memory able to retain the digital information it stores even when not powered. The most widely used nowadays are Flash memories, in particular the NAND type.

**Recursive algorithms:** a recursive algorithm is one that contains a recursive function, i.e. a function that calls itself within its code.
SECTION 1: STEP RESPONSE OF FIRST ORDER LINEAR SYSTEMS

Introduction: a linear system is a system whose output is linearly dependent on the input. We will restrict our analysis to the sole linear systems whose components are all passive and linear (resistors, capacitors and/or inductors). The number of reactive components of such systems (capacitors and inductors) determines the order of the system: a linear system with only one reactive component (a capacitor or an inductor) is a first-order linear system; if the reactive components of the systems are two (one inductor and one capacitor, or two capacitors, or two inductors) the system is a second-order one. Linear systems have several applications on Electronics and Telecommunications: they can be used to filter electronic signals, to minimize noise, to demodulate and delay signals. In general, they represent a fundamental block for the elaboration of analog electronic signals. Even if in actual electronic and telecommunication systems filters and modems are not made of only passive components, the analysis of first and second order passive linear systems is important for two reasons: it is simple and it features all the important characteristics of a more general linear system. So we can easily study the most important characteristics of a first or second-order linear system, understand its behavior and its limits, and next move forward to the analysis of more performant but more complex linear systems, too.

The problem: in this Section we will analyze the step response of the first order linear system of Figure 1:

![Figure 1](image-url)

**Figure 1:** the input voltage $V_I(t)$ of the L-R system is a step waveform (i.e. 0V for $t<0$ and 5V for $t \geq 0$). The output $V_O(t)$ is the voltage across the resistance. The problem we want to solve is to determine the shape of the output voltage waveform and show it on a graph, diagram or plot.

There are several ways to determine the shape of the output voltage waveform $V_O(t)$. First of all, for $t<0$ $V_I(t)$ is constant and equal to 0. When $V_I(t)$ is constant the inductor behaves as a short circuit, hence the voltage drop across the inductor $V_L(t) = 0$. So for $t<0$ $V_O(t)=V_I(t)-V_L(t)=V_I(t)=0$. For $t>0$ $V_I(t)$ is still constant but equal to 5V. Hence $V_O(t)$ is expected, after an initial transient whose duration depends on the time constant of the system $\tau=L/R=0.5\mu s$, to become equal to $V_I(t)=5V$. So what we really don’t know is only the transient response of $V_O(t)$ to the step voltage $V_I(t)$: how long does this transient last? What is the shape of the $V_O(t)$ waveform like during this transient?

---

1 By restricting our classification of electronic components to resistors, capacitors, inductors, diodes and transistors, we have: the resistor is a passive linear not reactive component; the capacitor and the inductor are reactive passive linear components; the diode is a non-linear passive component; the transistor is a non-linear active component, because its output current or voltage can be amplified with respect to its input value [1].
Figure 2: the output voltage waveform of our L-R system is =0V for t<0 and =5V for t≥t_{TR}, where t_{TR} is the duration of the transient, dependent on the time constant τ=L/R of the system. Our problem is to determine the value of t_{TR} and the shape of V_o(t) during the transient (between 0s and t_{TR}).

The most straightforward way to solve our problem is to directly solve the first-order differential equation describing the system [2]. However, we cannot use this method because we have not studied the derivatives and the mathematical theory behind differential equations yet. So we must rely on the two indirect ways we have studied during the course:

1. Finite difference time domain solution of the differential equation describing the problem: by dividing the time scale into several time intervals of the same width Δt (chosen significantly lower than the time constant of the system), we can approximate the rigorous differential equation solution of the problem into a system of algebraic equations that can be easily solved [3].

2. Solution of the system in the Laplace domain: the original circuit components and input voltage are transformed in the Laplace domain, obtaining a new circuit that is solved by an easy algebraic equation. The solution of the system in the Laplace domain is then inverse-transformed into the time domain, obtaining the desired output waveform [4].

Let’s see how we can implement both solutions by means of a C program.

**Finite difference method – numerical analysis:** our objective is to find the transient response of the output voltage V_o(t) to a step input voltage V_i(t) in the circuit of figure 1. By applying Ohm’s law to the resistor we obtain:

**Eq. 1.1:** \( V_o(t) = R \cdot I(t) \), where I(t) is the current flowing in the circuit.

On the other hand, by applying the generalized Ohm’s law to the inductor we have:

**Eq. 1.2:** \( L \cdot \frac{dI(t)}{dt} = V_i(t) - V_o(t) \), where \( \frac{dI(t)}{dt} \) is the derivative of the current. By substituting Eq. 1.1 into Eq. 1.2 and rearranging all the terms we obtain the first-order differential equation:
Eq. 1.3: \[ \frac{dI(t)}{dt} + \frac{1}{\tau} \cdot I(t) = \frac{5V}{L}, \] for t\geq0 with the initial condition I(t=0)=0A. \( \tau = \frac{L}{R} = 0.5\mu s \) is the time constant of the system.

Now we are blocked, because we don’t know how to solve the equation (it is not difficult to solve a linear differential equation as that in Eq. 1.3, but simply it is not the objective of this module). We now apply the finite difference approach to bypass our difficulty [3]. This is done by dividing the time scale into several intervals (or steps) of width (or stepsize) \( \Delta t = t_{n+1} - t_n \), where \( t_n = n\Delta t \) is the \( n \)th time instant. If \( \Delta t \) is sufficiently small we can approximate the current derivative \( \frac{dI(t)}{dt} \) calculated at the instant \( t_n \) with the incremental ratio 
\[ \frac{\Delta I_n}{\Delta t} = \frac{I_{n+1} - I_n}{t_{n+1} - t_n}. \]

Eq. 1.4: 
\[ \frac{\Delta I_n}{\Delta t} + \frac{1}{\tau} \cdot I_n = \frac{5V}{L}. \]

By multiplying Eq. 1.4 by the step interval \( \Delta t \) and rearranging it we get:

Eq. 1.5: 
\[ \Delta I_n = \Delta t \cdot \left( \frac{5V}{L} - \frac{I_n}{\tau} \right), \]
that gives the algebraic formulas for the change in the current when the time is stepped by one stepsize \( \Delta t \). When the stepsize \( \Delta t \) is very small, a good approximation to the underlying differential equation is achieved. By explicating Eq. 1.5 for the current at instant \( t_{n+1} \), i.e. \( I_{n+1} \), we find the formula according to Euler method:

Eq. 1.6: 
\[ I_{n+1} = I_n + \Delta t \cdot \left( \frac{5V}{L} - \frac{I_n}{\tau} \right). \]

Eq. 1.6 can be used to calculate \( I_1 \): in fact, \( I_0 = 0 \) is known, \( \Delta t \) is chosen by the programmer hence it is known, \( L \) and \( \tau \) are known and dependent only on the systems components. Once \( I_1 \) is known, Eq. 1.6 can be used to calculate \( I_2 \), and so on until we are able to determine all the discrete current values starting from the first (with index 0) up to the \( n \)th. Once we have determined the discrete current values, we can easily calculate the discrete finite output voltage values \( V_{On} = R \cdot I_n \) and plot them versus time obtaining the desired step transient response of our system.

**C programming solution – input variables definition:** Both a C program and an Excel spreadsheet can be used to implement and solve Eq. 1.6. In this paper we are interested in the C programming solution. First of all, we need to identify all the input variables of our problem. What do we know? We know the input voltage waveform \( V_f(t) \), and the value of the systems components \( L \) (500\( \mu \)H) and \( R \) (1K\( \Omega \)). With regard to \( V_f(t) \) we’re interested in its initial (at t=0s) and final (t>\( t_{TR} \)) values: \( V_{ini} = 0 \) and \( V_{fin}=5V \). From the initial and final input voltage values, the initial and final current values can be calculated: \( I_{ini}=V_{ini}/R=0A \) and \( I_{fin}=V_{fin}/R=5mA \). The last input data we need to start to solve the problem is the stepsize \( \Delta t \) and the number N of steps we want to calculate. Both \( \Delta t \) and N are parameters that need to be fixed by the programmer and whose choice will

---

2 In practice, we divide the time axis into several steps of width \( \Delta t \) obtaining a number of discrete time instants \( t_0=0, t_1=\Delta t, t_2=2\Delta t, ... t_n=n\Delta t \). A discrete current value \( I_0=0A, I_1=? , I_2=?, ..., I_n=? \) corresponds to each discrete time instant. Our problem is to determine the current values at each discrete time instant. The value of \( \Delta t \) is decided by the programmer: this choice is crucial because it determines the overall accuracy of the finite difference method.
significantly affect the accuracy of the solution provided by the program. In particular $N$ and $\Delta t$ should be chosen so as to satisfy:

Eq. 1.7: $\Delta t << \tau$

$$N \cdot \Delta t \geq t_{TR},$$

The first condition means the stepsize should be significantly lower than the time constant of the system: this condition is required to guarantee the accuracy of the finite difference solution (minimize the error); the second condition means the number $N$ of time steps should be large enough to cover the whole transient of the output voltage, whose duration is $t_{TR}$: the meaning of this condition is obvious.

What is the best choice for $\Delta t$ and $N$? Remember that $N$ is the number of discrete current values we need to calculate: the higher $N$, the higher the memory space the program needs to store all the current values ($I_0$, $I_1$, $I_2$, ..., $I_N$) and the number of mathematical operations it needs to run. Program time and memory allocation are not optimized. Therefore we’ll choose $N$ as the lowest integer greater than $\frac{t_{TR}}{\Delta t}$: equation 1.7b is thus satisfied and the numbers of current values to store and calculate is minimized. Now we need to choose $\Delta t$: values lower than $\tau/10$ are recommended in order to provide a sufficiently accurate solution. On the other hand, we should avoid values lower than $\tau/100$ because they would result in high values of $N$ and hence a not efficient program both from the viewpoint of execution time and memory occupation. For those who remember the theory of first-order linear systems $t_{TR}$ is not an unknown quantity, since it can be easily approximated with $5\tau$ (after 5 time constant since the beginning of the transient the system is for all practical purposes in a steady state condition). Therefore, if we choose $\Delta t=\tau/10$ we’ll get $N=50$, while if we choose $\Delta t=\tau/100$ we’ll get $N=500$, which are both reasonable values. We’ll investigate after the program is ready how the choice of $\Delta t$ affects solution accuracy: we’ll start to decrease $\Delta t$ from $\tau/10$ down to $\tau/100$ and compare the accuracy of the different solutions for each value of $\Delta t$. We’ll see there is a threshold below which decreasing $\Delta t$ further does not provide significant advantages in terms of solution accuracy and hence it’s not efficient. In conclusion, the input variables of our program are:

1. $V_{ini} \rightarrow$ used to calculate $I_{ini}=I_0$, the initial solution from which the calculation starts;
2. $V_{fin} \rightarrow$ to be used in Eq. 1.6 instead of 5V;
3. L and R values $\rightarrow$ to be used to calculate $\tau=L/R$, that appears in Eq. 1.6 along with L value;
4. $t_{TR}$ and $\Delta t$ $\rightarrow$ to be used to calculate the number of time steps $N=\frac{t_{TR}}{\Delta t}$; $\Delta t$ also appears in Eq. 1.6.

The program will ask the user to insert $V_{ini}$, $V_{fin}$, and $\Delta t$ in sequence. L, R and $t_{TR}$ will be defined within the program by the programmer: they won’t be asked the user. Note all these values are real quantities, so they should be defined as *float* or *double*. A 1mV=$10^{-3}$V calculation accuracy is enough for us, so we can rely on the least memory consuming *float* data type. The measurement unit of these quantities will be specified to the user in order to avoid program errors. For instance, the
value of the stepsize $\Delta t$ is 50E-9s or 50ns. The program has to specify to the user if the stepsize has to be inserted in s (50E-9) or, instead, in ns (50).

**C programming solution – the program outputs:** What is the expected output of our program? The final result we want to reach is a plot of $V_O(t)$ versus time. Using Eq. 1.6 our program is able to calculate all the discrete current values $I_0, I_1, \ldots, I_N$, that could be stored in an array of $N$ float elements. It is sufficient to multiply each of this array elements by $R$ to obtain the discrete output voltage values $V_{00}, V_{01}, \ldots, V_{ON}$. So the output of our program is a file made of two columns: the first column contains the discrete time instants $t_0, t_1, \ldots, t_N$ (i.e. the time axis of the plot) while in the second column there are the corresponding output voltage values (the voltage axis of the plot), as shown in Figure 3. We want to store the output voltage versus time in a file in order to subsequently process it with a program able to graphically visualize the corresponding plot. The name of the file is chosen by the programmer and displayed on the PC screen so as to be visible to the user; the file will be stored in the same directory of the executable code of the program.

<table>
<thead>
<tr>
<th>time axis [s]</th>
<th>voltage axis [V]</th>
</tr>
</thead>
<tbody>
<tr>
<td>$t_0$</td>
<td>$V_{00}$</td>
</tr>
<tr>
<td>$t_1$</td>
<td>$V_{01}$</td>
</tr>
<tr>
<td>$\ldots$</td>
<td>$\ldots$</td>
</tr>
<tr>
<td>$t_N$</td>
<td>$V_{ON}$</td>
</tr>
</tbody>
</table>

**Figure 3:** example of the content of the file generated by the C program solving the linear system by means of finite difference analysis. This file may be subsequently read by other programs able to draw the corresponding plot (for instance MS Excel).

Therefore, the program will define an array of $N$ float elements where to store the discrete current values and a FILE pointer$^3$ where to save the voltage versus time values for post-processing with Excel or other graphical tools.

A final important note regards the value of $N$: it depends on the input variables $t_{TR}$ and $\Delta t$ inserted by the user. However, program memory allocation occurs before the execution of the program. So, how can we allocate an array of $N$ float elements if we do not know how much $N$ is? We’ll assume a maximum value of $N$ equal to 1000 and allocate an array of 1000 elements accordingly. If the value of $\frac{t_{TR}}{\Delta t}$ provided by the user is $\leq 1000$ there are no issues: we won’t use all the elements of the array – our program is not efficient because we have allocated more memory than the one required. On the contrary when $\frac{t_{TR}}{\Delta t} > 1000$ we will assume $N=1000$, so we will use all the elements of the array: in this case, however, the program should show the user that the actual value of $t_{TR}$ is not the one inserted but it is $1000\Delta t$, i.e. lower than the one inserted. This means that maybe the condition of Eq. 1.7b is not met and we won’t be able to cover the whole transient response of the system. If this is the case, the only solution is to restart the program with a higher value for $\Delta t$.

---

$^3$ A FILE pointer is a memory address reserved by the program for the data contained in the file.
C programming solution – algorithm: Figure 4 shows the flowchart of the most simple algorithm we can implement to calculate the N discrete current values and print file the desired discrete values of output voltage in the output file. The algorithm contains only one iterative control structure, namely a for cycle with index j running from 0 to N. At each cycle the j-th output voltage value is already known and printed in the output file; the next (j+1)-th current value is then calculated using Eq. 1.6; finally j is incremented until it becomes greater than N. When j > N the program stops and the output file contains all the data we need to build our plot of $V_o(t)$ versus time t.

Figure 5b shows an ANSI-C implementation of the flowchart. Note the name of the output file is decided by the user and we recommend to use file names with extension .txt (text file), for instance lrtran.txt as in Figure 5a, that will be automatically opened by Notepad application.

Figure 4: simplified flowchart of the proposed algorithm for the calculation of the N discrete current values. The algorithm prints the desired discrete output voltage waveform into the user specified output file.
Figure 5a: MS-DOS command prompt showing the interaction between the program and the user.

```c
#include <stdio.h>

#define L 500E-6 // 500uH inductor
#define R 1E3 // 1KOhm resistor
#define tau (L/R) // time constant of the system
#define tTR (5*tau) // duration of the transient fixed at 5 time constant

int main ()
{
    FILE *fp; // pointer to the output file
    unsigned int j, N;
    char risp, filename[10];
    float VIi, VIf, Dt, I[1000];

    printf("Please insert step waveform initial voltage in V (0 default): ");
    risp=scanf("%f", &VIi);
    if (!risp) VIi=0;
    printf("Step waveform initial voltage is %.2fV
"),VIi);
    printf("Please insert step waveform final voltage in V (5 default): ");
    risp=scanf("%f", &VIf);
    if (!risp) VIf=5;
    printf("Step waveform final voltage is %.2fV\n",VIf);
    printf("Please insert finite difference analysis stepsize in s (%g default): ",((L/R)/10));
    risp=scanf("%f", &Dt);
    if (!risp) Dt=(L/R)/10;
    printf("Finite difference analysis stepsize is %.3g \n",Dt);
    scanf("%s",filename);
    fp=fopen(filename, "w"); // open the file in write mode
    fprintf(fp,"## L-R output voltage waveform\n");
    fprintf(fp,"## problem inputs: VIi=%.2fV, VIf=%.2fV, Dt=%gs, L=%gH, R=%gOhm\n",VIi,VIf,Dt,L,R);
    fprintf(fp,"##\n");
    i=0; j=0; N=N+1000;
    printf("\nWarning: Dt is too small, I'll calculate transient response only up to %gs\n",N*Dt);
}
```
printf("\nPress any key to exit... ");
getchar();
}

**Figure 5b:** code of the ANSI-C program implementing the finite difference analysis of the L-R system.

The output of the C program is a text file with the name given by the user during program execution that can be opened by Notepad or Wordpad applications. Figure 6 shows the first few lines of the file:

```plaintext
## L-R output voltage waveform
## problem inputs: VIi=0.00V, VIf=5.00V, Dt=1.25e-008s, L=0.0005H, R=10000hm
##
t[s]   VO[V]   IO[A]
0.000e+000 0.000 0
1.250e-008 0.125 0.000125
2.500e-008 0.247 0.000247
3.750e-008 0.366 0.000366
5.000e-008 0.482 0.000482
6.250e-008 0.595 0.000595
7.500e-008 0.705 0.000705
......
```

**Figure 6:** the output of the program is a text file of 3 columns: in the first column there are the discrete time instants (in s), the second and third columns contain the corresponding discrete output voltage (in V) and current values (in A).

**C programming solution – graphical representation:** what do we do with the output file? We are interested in the output voltage waveform behavior. So we need a way to plot the discrete voltage values versus time. The Get External Data feature of MS-Excel 2007 can be used to this purpose: selecting Data → Get External Data → From Text allows to import all the data contained in a specified text file within an Excel worksheet. It is sufficient to choose the correct delimiter (in our case Tab, the tabulation), select General as column data format and, finally, select Advanced to choose the decimal separator (MS Excel default is the comma) and the thousand separator (MS Excel default is the dot). Since for C programs the dot (.) is the decimal separator, we need to change Excel settings accordingly in order to correctly read the output data file. In fact, the decimal separator is the dot in our text file, while the thousand separator is the comma. Figure 7 shows how the MS-Excel worksheet appears after the importation of the output file data is finished.

![Figure 7](image.png)

**Figure 7:** the same data shown in Fig. 6 after they have been imported into MS-Excel. Note that the dot separator of the original text file of Fig. 6 has been replaced by the comma separator in the Excel file, thanks to the Advanced setting modification performed during the importation of the data.
Now that output voltage and current values versus time are imported in Excel it is very easy to draw a plot: select only the numerical values of the first two columns (starting from row 5 to the end) and then click on Insert → Charts → Scatter. The plot of Figure 8 shows what we are looking for: the transient behavior of the output voltage of the L-R system. As expected it is exponential with time constant $\tau=L/R=0.5\mu s$. After 5 time constants the transient is practically finished and the output voltage has reached its final constant value of 5V. Note that $V_0$ waveform seems really exponential, while we know for sure it is an optical effect produced by the union of several linear waveforms when the stepsize is small.

![L-R transient response](image)

**Figure 8:** transient response of the L-R system as calculated by a C program implementing the finite difference method.

**Considerations on the C implementation of the finite difference method:** the finite difference method we have used to solve the L-R transient is basically the same used in all numerical simulators. In particular, the circuit simulator PSPICE, which is the basis of Electronic Workbench, LTspice, and other proprietary C.A.D. (Computer Aided Design) tools, solves the partial integro-differential equations that describe a circuit by finite element method, a more complex version of the finite difference method we have used [5]. In fact, the practical method used to transform the partial differential equations into algebraic ones is not as simple as the Euler’s formula we have used. In actual numerical simulators the stepsize is adaptively modified during calculations in order to achieve a specified accuracy: the choice of the stepsize made by the user is not as critical as in our algorithm. The stepsize is also not constant: in the time intervals where the waveform to be simulated exhibits the sharpest changes, the stepsizes are very small, while where the waveform variations are small the stepsizes are larger. Examples of C programs implementing finite element analysis to solve ordinary differential equations able to achieve a specified accuracy and to adapt the stepsize can be found in [6]. Here we will show the result of the circuit simulation of our LR system obtained with the CAD tool LTspiceIV. LT stands for Linear Technology, a U.S. semiconductor company that developed this PSPICE version. LTspiceIV is a freeware software for circuit simulation that can be downloaded from the Linear Technology Company site [7] and is able to perform DC, AC and transient analysis of linear and non-linear circuits. Figure 9 shows the result of the simulation. It is clear from the picture that LTspice has divided the simulation time into several
intervals of different widths (it applies an adaptive algorithm) within each of which the solution is linear.

Figure 9: transient response of the LR system as simulated by LTspiceIV. The waveform is similar to the one obtained with our C program. The zoom around 1us clearly shows the output waveform is the union of linear waveforms resulting from the subdivision of the simulation time into several steps of finite size.

C programming solution – questions and exercises:

1) What is the crucial parameter that determines the accuracy of a finite difference solution of an ordinary differential equation?

2) Run the C program that calculates the transient response of the L-R system for three different values of the stepsize: \( \tau/2 \), \( \tau/10 \) and \( \tau/50 \). Explain if and why the three solutions are different. What’s the most accurate one?

3) Try to change the directive \#define tau (L/R) with the directive \#define tau L/R (i.e. rewrite the directive without parentheses). The program stops working correctly. Can you explain why?

4) The C program of Figure 5b) asks the user to insert the values of the initial and final input voltage values, Vli and Vlf. Can you explain why in Fig. 5a) the user digits the character ‘q’ instead of a voltage value and why the program reads 0V as Vli and 5V as Vlf?

5) I want to find the transient response of the same circuit but with different values of L and R, for instance L=100\( \mu \)H and R=5K\( \Omega \). What should I change in the C program?

6) Try to run the C program for different values of Vli and Vlf, for instance Vli=8V and Vlf=2V. What is the behavior of the output voltage waveform? Is it still correct?
7) In Eq. 1.6 $I_{n+1}$ depends only on $I_n$ and other known quantities. A recursive algorithm is therefore possible. What code lines should be changed to implement the recursive algorithm? And what are the pros. and cons. of the recursive solution?

**Laplace domain solution – the symbolic approach:** in this section we want to find the transient step response of the system of Figure 1 by means of the Laplace transform method [4,8]. The first step is to draw the original circuit of Figure 1 in the Laplace domain considering that:

- The input voltage $V_I(t)=5V*u(t)$ ($u(t)$ is the step function, 0 for $t<0$ and 1 for $t\geq 0$) becomes $V_I(s)=5V/s$;
- The inductor $L=500\mu H$ becomes the series of the symbolic impedance $Z_L=s*L$ and the voltage generator $-L*I_0$, where $I_0$ is the initial current value;
- The resistor remains the same as in the time domain.

Hence we obtain the circuit of Figure 10 in the Laplace domain:

![Figure 10: the circuit of Figure 1 transformed into the Laplace domain.](image)

By applying generalized Ohm’s law and Kirchoff’s principles to solve this simple circuit, it is very easy to get:

**Eq. 1.8:**

$$V_o(s) = R5 \cdot \left( \frac{s}{R5 + sL} \right) = \frac{5V}{\tau} \cdot \frac{1}{s \cdot (s + \frac{1}{\tau})}$$

assuming $I(0)=0$ (initial current value in the inductor is 0). $\tau$ is of course the time constant of the circuit: $\tau = L/R5$.

The inverse transform of $1/(s(s+a))$ is $(1-e^{-at})/a$, as can be derived by the Laplace Transform Table provided in Appendix A or in any electronic textbook [4,9]. Therefore, the inverse transform of Eq. 1.8 is:

**Eq. 1.9:**

$$V_o(t) = \frac{5V}{\tau} \cdot \left( 1 \cdot e^{-\frac{t}{\tau}} \right) = 5V \cdot \left( 1 - e^{-\frac{t}{\tau}} \right),$$

that is the expected exponential inductor charging waveform.

**Laplace domain solution – the C program:** in this section we are going to write a C program that implements equation 1.9 when using *math.h*, the mathematical library available in C. By using the
same approach as in the first program, the only input variable of this new program is VIf, the final input voltage value. In fact, Eq. 1.8 and 1.9 hold only if I(0)=0; but I(0)=0 only if VIi=0. L and R are the input parameters of the circuit. The output of the program is the output voltage \( V_O(t) \) that we want to plot versus time. The duration of the transient and hence the last instant to be plotted is already known and equal to \( 5*\tau \), dependent only on the parameters of the circuit. However, if we need a plot of \( V_O(t) \) vs. \( t \), for each time instant of interest it is necessary to write the corresponding output voltage value in a file, as we did in the previous case. So, even in this program we need to define a time step \( Dt \) that represents the distance between two consecutive time instants. However, in this case the choice of \( Dt \) does not affect the accuracy of the output voltage values, because Eq. 1.9 is the exact solution of the step response of the system. The choice of \( Dt \) only affects the appearance of \( V_O(t) \) vs. \( t \) graph: if \( Dt \) is chosen too large, the graph will appear as a collection of straight lines because there isn’t a sufficient number of points. Since the only concern is the appearance of \( V_O(t) \) vs. \( t \) graph, in this program we will use the number \( N \) of data points instead of \( Dt \) as input variable. Usually, with \( N=100 \) data points any transient plot is sufficiently well defined; when this is not the case, the user simply needs to re-run the program with a higher value of \( N \), without caring about stepsize \( Dt \) that will be internally calculated by the program. Figure 11 shows the proposed C code implementing Eq. 1.9:

```c
/* LR step response - LRlapl.cpp */

This program implements the L-R inductor charging transient.
Data inputs (user defined): input waveform final voltage value VIf (in V), and the number of data points to be plotted, \( N \)
Other inputs (program defined): component values, L (in H) and R (in Ohm)
Data elaboration: the program calculates the output voltage value at each specified data points
Data Outputs: file containing \( N \) data points reporting output voltage versus time */

// ANSI-C libraries included by the program
#include <stdio.h>
#include <math.h>

// program defined input data
#define L 500E-6  // 500uH inductor
#define R 1E3    // 1KOhm resistor
#define tau (L/R) // time constant of the system
#define tTR (5*tau) // duration of the transient fixed at 5 time constant

// program variables definition
FILE *fp;
unsigned int j, N;
char risp, filename[10];
float VIf;

int main ()
{
    // reading of user defined input data
    printf("Please insert step waveform final voltage in V (5 default): ");
    risp=scanf("%f", &VIf);
    fflush(stdin);
    if ( !risp ) VIf=5;
    printf("Step waveform final voltage is %.2fV\n", VIf);
    printf("Please insert the number of data points to be plotted (100 default): ");
    risp=scanf("%d", &N);
    fflush(stdin);
    if ( !risp || (N<=1) || N>100,
    printf("The number of data points in %d\n",N));
    scanf("%s", filename);
    fflush(stdin);
    printf("Press any key to start output file writing...");
    getchar();
    fflush(stdin);
    fp=fopen(filename, "w"); // open the file in write mode
    fprintf(fp,"## L-R output voltage waveform\n");
    fprintf(fp,"## problem inputs: VIi=0V, VIf=%.2fV, L=%gH, R=%gOhm\n",VIf,L,R);
    fprintf(fp,"##\t\t\t\t\t\t\t\t\t\n");
    // calculate output voltage and write to output file
    for (j=0; j<N; j++) fprintf(fp,"\t%.3e\t%.3f\n",j*tTR/N,VIf*(1-exp(-j*tTR/N)/tau));
}*/
```
// close the output file
fclose(fp);
printf("\nPress any key to exit... ");
getchar();
}

**Figure 11:** code of the ANSI-C program implementing Eq. 1.9, the exact solution of the L-R step response.

Figure 12 shows the result of the program of Fig. 11 after its output file has been imported into an Excel sheet using the same procedure described above. As it can be seen, 100 data points are sufficient to provide a smooth waveform. When Fig. 8 and 12 are compared it seems there is no difference. Actually, there is a difference between the graph of Fig. 8 and 12 whose magnitude depends on the choice of the stepsize $\Delta t$ done to obtain Fig. 8: the smaller the stepsize $\Delta t$, the smaller the difference between the two plots of Fig. 8 and Fig. 12 (the better the finite difference approximation is). Since smaller stepsize means higher finite difference method computational time, a trade-off between numerical approach accuracy and computational time is in order.

![Figure 12: exact solution of the transient response of the L-R system represented with 100 data points.](image)

The **trade-off between computational time and accuracy**: to investigate the trade-off between computational time and accuracy in the finite difference approach we need to define what we exactly mean by **accuracy**. Accuracy is the percentage error between the numerical solution of the problem and the exact solution. What does this mean? Let’s consider the $j$-th discrete time instant $t_j = j * \Delta t$: the finite difference approach provides an output voltage value at this instant equal to $V_{OFE}[j]$; the exact solution, calculated for instance by using the result of the Laplace transform method, provides a different value at the same instant equal to $V_{OLT}[j]$. The percentage error of the finite difference approach at the time instant $t_j = j * \Delta t$ is therefore given by:

**Eq. 1.10:**

$$Err_j = \frac{V_{OLT}[j] - V_{OFE}[j]}{V_{OLT}[j]} \cdot 100 = \frac{I_{LT}[j] - I_{EF}[j]}{I_{LT}[j]} \cdot 100.$$  

Note that the percentage error of output voltage is the same as that of circuit current, since $V_o = R * I$. The accuracy of the finite difference approach is defined as $\text{Max}\{Err_j\}$ for $j = 0$ to $j = N$, where $N$ is the number of data points. In practice it is the maximum percentage difference between the exact solution and the numerical one considering all the discrete time instants.

The investigation of the trade-off between accuracy and the stepsize $\Delta t$ (that is inversely proportional to computational time) can be performed by means of a C program whose output is a text file reporting for each time instant $t_j$ the corresponding value of $Err_j$. By running this program...
for several different values of \(Dt\) and comparing the resulting \(Err_j\) vs. \(t_j\) plots we may get an immediate visual result of the trade-off between accuracy and computational time.

The inputs of the program are: \(VI_i, VIf, Dt\); the input parameters are \(L\) and \(R\); the program is a combination of the first two programs because it needs to calculate both the finite difference solution and the exact exponential one. After finding the two solutions, it implements Eq. 1.10 to find the percentage error at each time instant and plot it versus time on the output file. Here follows the C code of the program:

```c
/*** analysis of LR finite difference solution – LRtradeoff.cpp ***/
This program calculates the percentage error of the finite difference solution of the step response of an L-R linear system.

Data inputs (user defined):
  - input waveform, \(VI_i\) and \(VIf\) – initial and final voltage values (in V)
  - finite element width (stepsize), \(Dt\) (in s)

Other inputs (program defined):
  - component values, \(L\) (in H) and \(R\) (in Ohm)

Data elaboration: the function \(cfe(j)\) calculates the finite difference current value at the discrete time instant \(t_j=j*Dt\), for \(j=0\) to \(j=N\)
  - function \(clt(j)\) calculates the exact current value at \(t_j=j*Dt\)

Data Outputs: file containing for each discrete time instant the corresponding percentage error of the finite difference solution \(Err_j\) for \(j=0\) to \(j=N\)

// ANSI-C libraries included by the program
#include <stdio.h>
#include <math.h>

// program defined input data
#define L 500E-6 // 500\mu H inductor
#define R 1E3 // 1K\Omega resistor
#define tau (L/R) // time constant of the system
#define tTR (5*tau) // duration of the transient fixed at 5 time constant

// functions declaration
float cfe(unsigned int n); // returns finite difference current value at instant \(tn=n*Dt\)
float clt(unsigned int n); // returns exact current value at instant \(tn=n*Dt\)

// program variables definition
FILE *fp;
unsigned int j, N;
char risp, filename[10];
float VIi, VIf, Dt, Ife, Ilt;

int main (){ // reading of user defined input data
  printf("Please insert step waveform initial voltage in V (0 default): ");
  risp=scanf("%f", &VIi);
  fflush(stdin);
  if (!risp) VIi=0;
  printf("Step waveform initial voltage is %.2fV\n", VIi);

  printf("Please insert step waveform final voltage in V (5 default): ");
  risp=scanf("%f", &VIf);
  fflush(stdin);
  if (!risp) VIf=5;
  printf("Step waveform final voltage is %.2fV\n", VIf);

  printf("Please insert finite difference analysis stepsize in s (%g default): ", (L/R)/10);
  risp=scanf("%f", &Dt);
  fflush(stdin);
  if (!risp || (Dt<0.0) || (Dt>(L/R)/10) ) Dt=(L/R)/10;
  printf("Finite difference analysis stepsize is %.3g s\n", Dt);
  scanf("%s", filename);
  fflush(stdin);
  printf("Press any key to start output file writing...\n");
  getchar();
  fflush(stdin);
  fp=fopen(filename, "w"); // open the file in write mode
  fprintf(fp,"##	 L-R output voltage waveform\n");
  fprintf(fp,"##	 problem inputs: VIi=%.2fV, VIf=%.2fV, Dt=%gs, L=%gH, R=%gOhm\n", VIi, VIf, Dt, L, R);
  fprintf(fp,"##	 outputs: percentage error Err, exact output voltage VOlt and numerical output voltage VOfe\n");
  fprintf(fp,"##	 time[t]	 Err[t]\t VOlt[V]\t VOfe[V]\n");

  // initialization
  N=tTR/Dt+1;
  if (N>1000){
    N=1000;
    printf("\nWarning: Dt is too small, I'll calculate transient response only up to %gs\n", N*Dt);
    }
  // calculate and write error to output file
```

16
fprintf(fp,"%.2e %.2f %.3f %.3f\n",0.0,VIi,VIi);
for (j=1; j<=N; j++)
    Ilt=clt(j), Ife=cfe(j); // current values stored in temporary variables
    if (Ilt==0) Ilt+=1e-18;
    fprintf(fp,"%.2e %.2f %.3f %.3f\n",j*Dt,100*(Ilt-Ife)/Ilt,R*Ilt,R*Ife);
} // close the output file
fclose(fp);
printf("nPress any key to exit... ");
getchar();
}

// functions body
float cfe(unsigned int n){ // returns the finite difference current value at instant tn=n*Dt
    if (!n) return (VIi/R); // if n==0 current is VIi/R
    else return (Dt*VIf/L+(1-Dt/tau)*cfe(n-1));
}
float clt(unsigned int n){ // returns the exact current value at instant tn=n*Dt
    return ((VIf+(VIi-VIf)*exp(-(n*Dt/tau)))/R);
}

Figure 13: code of the ANSI-C program that calculates the percentage error of the finite difference solution.

The program stores both the finite difference and exact solution in two temporary variables, Ife and Ilt respectively, in order to minimize the number of function calls and hence computational time (note cfe(n) is a recursive function time and memory consuming). We can also write both the finite difference and exact output voltage in the output file with minimum calculation overhead. Note the initial instant is written in the output file before the for cycle: in fact, both finite difference and Laplace method provide the same solution at t=0 (the output voltage equals VIi), so the error is 0. This avoids a calculation overflow when VIi=0V (hence Ilt=clt(0)=0) due to the division by zero in the error expression 100*(Ilt-Ife)/Ilt. The program works correctly even for VIi or VIf < 0 and it is able to simulate both inductor charge and discharge. To account for these cases, the program checks a possible 0 value for Ilt that would give rise to a division by zero and hence a calculation overflow. When Ilt is zero the program assigns it a value close to zero, i.e. 1e-18, just to avoid the division by zero.

We run the C program of Fig. 13 with input data VIi=0, VIf=5V and for three different values of Dt: Dt=50ns (5e-8s=τ/10), Dt=20ns (2e-8s= τ/25) e Dt=10ns (1e-8s=τ/50). The three plots of Errj vs. time are reported in Fig. 14 showing the trade-off between Dt value (inversely proportional to computational time) and accuracy of the finite difference solution.

Figure 14: visual representation of the trade-off between stepsize Dt (inversely proportional to computational time) and accuracy (percentage error) of the finite difference solution.
Fig. 14 shows that the accuracy is about -5% with a \( \tau/10 \) (50ns) stepsize; the accuracy improves to about -2% with a \( \tau/25 \) (20ns) stepsize. To reach a -1% accuracy, however, stepsize should be as small as \( \tau/50 \) (10ns), that corresponds to dividing the transient into 250 data points. Note the percentage error is negative because the finite difference solution provides current and voltage values higher than the exact ones. Also note the percentage error is high during the initial phase of the transient, where current/voltage values are close to zero and their variation with time is high: towards the end of the transient the percentage error is within -0.2% whatever stepsize is chosen, because final current/voltage values are high and their variation with time is small.

C programming Laplace method solution – questions and exercises:

1) Write a C program able to plot the Laplace solution of the circuit of Fig. 10 when \( I(0) \) is different from 0A. (Suggestion: first derive the exact mathematical resolution formula by using Laplace transform table of Appendix A, and then implement it with a C code).

2) Run the program of Fig. 13 with \( VI_i > VI_f \), by simulating an inductor discharge. What is the sign of the finite difference percentage error and where is the highest percentage error found? Also try to use a negative voltage value for \( VI_i \) and/or \( VI_f \) and discuss the behavior of the corresponding percentage error.

3) Modify the C code of Fig. 13 in order to find the maximum \( Dt \) value able to guarantee a given input accuracy. In practice, the program input is the desired level of accuracy (for instance \( \pm 0.5\% \)) and the output is the maximum \( Dt \) value able to guarantee the specified accuracy.

4) Write a C code similar to that of Fig. 13 but whose objective is the evaluation of the absolute error of the finite difference method, i.e. \( Er_A[j]=V_{OLT}[j]-V_{OFF}[j] \). Then run the new program with the same inputs as in Fig. 14 and for the same three different values of \( Dt \) of Fig. 14, namely \( Dt=\tau/10 \) (50ns), \( Dt=\tau/25 \) (20ns) and \( Dt=\tau/50 \) (10ns); finally plot the resulting absolute errors versus time as in Fig. 14. What is the difference between absolute and percentage errors? Explain the differences between the plots of absolute and percentage errors versus time.

5) Write the C code to calculate the step response of the following first-order linear systems by using the Laplace transform or the finite difference method: R-L system, R-C system, C-R system.

SECTION 2: SOME ODD RESULTS ON LINEAR SYSTEMS

Introduction: in this Section we’ll analyse the behavior of the transient response of some particular linear systems with the Laplace method. At the end of the Section we’ll summarize all the linear systems that can be easily analyzed by the Laplace transform method providing a C library containing all the functions that reproduce their transient response. We’ll show some examples on how to use this library and suggest some exercises.
Step response of the LC system: the order of a passive linear system is the same as the number of reactive components (capacitor, inductor) present in the system. The L-R system of Fig. 1 contains only an inductor and hence it is a first order linear system. The L-C system of Fig. 15 contains an inductor and a capacitor, hence it is a second order linear system. We can calculate the step response of a second order linear system by means of the Laplace transform method, by finding the exact output voltage waveform, or we may use the numerical finite difference approach and find an approximate waveform. In this Section we’ll use the Laplace transform method to solve the circuits. The first step is to draw the circuit in the Laplace domain. \( V_I(t) = V_I u(t) \) is the step input voltage, whose final value is \( V_I \). Since \( V_I(t) = 0 \) for \( t \leq 0 \) both the current and the voltage across the capacitor are 0 when \( t=0^+ \) (zero initial condition).

**Figure 15:** the input voltage \( V_I(t) \) of the L-C second order linear system is a step waveform (i.e. 0V for \( t<0 \) and 5V for \( t \geq 0 \)). The output \( V_O(t) \) is the voltage across the capacitor C. The problem to be solved is to determine the shape of the output voltage waveform and show it on a graph.

By following these considerations we can draw the circuit in the Laplace domain in Figure 16:

**Figure 16:** Representation of the circuit of Fig. 15 in the Laplace domain with step input \( V_I * u(t) \).

The second step is to solve the circuit in the Laplace domain by using Ohm’s and Kirchhoff’s laws:

\[
V_O(s) = \frac{1}{s \cdot C} \cdot I(s) = \frac{1}{s \cdot C} \cdot \frac{V_I}{s} = \frac{V_I}{s^2 \cdot L \cdot C + 1} = \frac{V_I}{s^2 \cdot \omega_n^2 + \omega_n^2},
\]

where \( \omega_n \) is the natural angular frequency of the system: \( \omega_n = \frac{1}{\sqrt{L \cdot C}} \). From the Laplace transform Table of Appendix A we know that the inverse transform of \( \frac{1}{s \cdot (s^2 + \omega_n^2)} \) is \( \frac{1 - \cos(\omega_n \cdot t)}{\omega_n^2} \). Hence the inverse transform of Eq. 2.1 is:
Eq. 2.2: \( V_o(t) = V_{If} \cdot \omega_n \cdot \frac{1 - \cos(\omega_n \cdot t)}{\omega_n^2} = V_{If} \cdot (1 - \cos(\omega_n \cdot t)) \),

whose waveform is shown in Figure 17.

The analysis shows that the step response of an LC circuit is a periodic waveform with period \( T=2\pi/\omega_n \) (\( T=62.8\mu s \) when \( L=1\text{mH} \) and \( C=100\text{nF} \) as in Fig. 17) whose average value is \( V_{If} \) (\( V_{If}=5\text{V} \) in Fig. 17). This is quite a different behavior from the step response of first order linear systems. First of all, each first order linear system is characterized by a time constant \( \tau \) and its transient response lasts for a time equal to \( 5*\tau \). After the transient is finished, i.e. for \( t\geq5*\tau \), the output voltage reaches the constant value equal to the circuit output voltage corresponding to a DC input equal to \( V_{If} \) (the final input voltage step value). In the LC system of Figure 15 there is no time constant but a natural angular frequency \( \omega_n \). The output voltage never reaches a constant value but it oscillates periodically around an average value equal to the circuit output voltage corresponding to a DC input equal to \( V_{If} \). This different behavior depends on the system stability, a characteristic that will be explored in the next paragraph.

**Correlation between step response and system stability:** the basic requirement of any automated analog control system is to be able to reach a steady state condition after the initial transient regulation phase is finished. In other words, at the turn on of any control system some oscillations are allowed, but they should not last long and the system should be able to reach regulation (steady state condition) as fast as possible. A robust analog control system should also feature the same capacity to keep regulation in front of the noises and disturbs present in the real world. Noise and disturbs are not predictable events that cause a variation from the regulation state of the system. The system should be able to go back to its original regulation state as soon as noise and disturbs stop. A system with these features is called stable [10,11].

A system is stable when the output corresponding to a bounded input is also bounded (BIBO stability, i.e. Bounded Input – Bounded Output stability). All first order linear systems analyzed in Section 1 are stable: in fact, their step response reaches a constant value after the initial \( 5*\tau \) transient. The LC system of Figure 15 is also stable, because its step response is also bounded even if it never reaches a constant value. However, the LC system of Figure 15 is only **marginally stable**, because its oscillations never stop in principle. A marginally stable system is not a safe
control system because it is not able to keep its regulation value. As an example, the simulation of Figure 18 shows the output voltage of an LC system biased at 5V subject to a disturb represented by a 1V pulse of 1µs width occurring at instant t=0.4ms.

![Figure 18: V(vo) is the output of the LC system when its input V(vi) is a 5V DC. At instant t=0.4ms a disturb of 1V height and 1µs width sums up and the output of the system starts to oscillate. The system is no more able to keep its regulation value.](image)

The disturb is able to produce oscillations at the natural angular frequency of the system and the stable regulation level is lost even if after the pulse V(vi) is still back to 5V.

The case of instability is even worse: in such a case a small disturb pulse is amplified and the output voltage starts to rise reaching the maximum voltage allowed by the system (system saturation) or damaging some components of the system itself. In both cases regulation is lost.

The step response of a system is a good way to understand its degree of stability:

1. are oscillations present but after a while does the system reach its constant output voltage steady state condition? The system is stable but its stability margins are not so wide. It’s better to improve its stability margins before using it as an actual control system.

2. are there no oscillations and does the system response resemble that of a first order linear system? The system is stable with good stability margins. It can be used as a control system.

3. do oscillations never stop but does the output voltage remain bounded? The system is marginally stable and it’s not suitable as a control/regulation system.

4. does output voltage rise up to its maximum allowed limit and remain fixed at it? The system is unstable and not suitable for control/regulation applications.

What about the lab? In actual life, marginally stable systems do not really exist. It’s better not to waste time to look for them. Just as an example try to build an LC circuit in your electronic lab. Program a step waveform from 0V to VIlf=5V on the pulse generator and look at the circuit output with an oscilloscope. The system behaves as in point 1 above, doesn’t it? It seems to be a stable system with a poor stability margin, as shown in Figure 19.

How is that possible? Laplace transform method clearly predicts a periodic waveform. The reason is quite simple: actual inductors and capacitors are not ideal components but they include parasitic components whose value should be specified in their Electrical Characteristics (EC) sheet [12,13]. In order to predict real systems behavior it is important to account for these parasitic effects in simulations.
Figure 19: $V(\text{vo})$ is the output of the LC system when its input $V(\text{vi})$ is a 0 to 5V step. The capacitor used in the LTspice simulation is a $1\mu\text{F}$ electrolytic capacitor from Nichicon (part number UPL1H010MAH) while the inductor is a 1mH inductor from Gowanda (p.n. 894AT1004V).

The CAD tool LTspiceIV allows us to choose among a wide set of real components, whose models include parasitics, specifying for each one the manufacturer and the part number. In particular, the main parasitic effect of a capacitor is its ESR (Equivalent Series Resistance) – the capacitor used in the simulation of Figure 19 has an ESR of $3.5\Omega$. The inductor of the simulation of Figure 19 includes a series resistance ($1.3\Omega$) a parallel resistance ($45K\Omega$) and a parallel capacitance ($220fF$), the most important of which is the $1.3\Omega$ series resistance. All things considered, when you build your LC circuit in the lab what you are really measuring is the following circuit, where $L$ and $C$ are now ideal components:

![Figure 20: LC equivalent circuit including the capacitor ESR RP2 and the inductor series resistance RP1. In this circuit $L$ and $C$ are really ideal components because their main parasitics are explicitly drawn. The voltage measured by the oscilloscope during the lab experiment is $V(\text{vo})$, the voltage drop across the series between RP2 and C.](image)

The real LC circuit is therefore an RLC series circuit, where the output voltage oscillations are dumped. The higher the values of the series resistances RP1 and RP2 the more damped the oscillations will be (i.e. the higher the damping factor $\zeta = \frac{R}{2 \sqrt{LC}}$). Since the RLC circuit is a stable one, after the step transient response is finished the circuit reaches a constant final voltage value equal to the output voltage of the system corresponding to a $V(\text{vi})$ DC bias (5V in the case of Figure 19).

Just as a conclusive note: the parasitic components of inductors and capacitors are not the only non-ideality of these components. In fact, there is another important non-ideality related to the dependence of inductance value on the current flowing through it and of capacitance value on the voltage across it. In practice, $L$ and $C$ values depend on $I_L$ (current flowing through the inductor) and $V_C$ (voltage across the capacitor) values, respectively. In particular $L$ and $C$ decrease as $I_L$ and
$V_C$ increase, respectively. It is important to be able to account for this effect when designing switching applications [14], such as filters for class D audio amplifier and switching regulators, where $I_L$ and $V_C$ variations may be significant.

**Some strange behavior of systems pulse response:** let’s now go back to a simple first order linear system, the RC circuit, to study its pulse response. The pulse response of a system is its output voltage waveform when a Dirac pulse is applied to the input. The Dirac pulse is a conceptual input voltage because it cannot be found in nature but it can only be approximated in the laboratory. It is indicated with the symbol $\delta(t)$ and it is equal to 0 for any value of time $t$ different from 0 ($\delta(t)$=0 for each $t \neq 0$) but the area below $\delta(t)$ waveform is $1V$s ($1$ Volt times second). In practice, it is a pulse whose value is $+\infty$ only when $t=0$ and equal to 0 for all other values of $t$.

The Laplace transform of $\delta(t)$ input is exactly equal to the area below $\delta(t)$ waveform, i.e. 1. So the RC circuit with input voltage $\delta(t)$ can be represented in the Laplace domain as shown in Figure 21.

![Figure 21](image)

**Figure 21:** RC circuit in the Laplace domain with Dirac pulse input voltage.

The circuit of Figure 21 is easily solved by using Ohm’s and Kirchoff’s laws:

**Eq. 2.3:** $V_o(s) = \frac{1}{s \cdot C} \cdot I(s) = \frac{1}{s \cdot C} \cdot \frac{1}{1 + \frac{1}{R \cdot s \cdot C}} = \frac{1}{\tau} \cdot \frac{1}{s + \frac{1}{\tau}}$.

As for any other first order linear system the time constant $\tau=R*C$ is defined. From the Laplace transform table of Appendix A $\frac{1}{s + \frac{1}{\tau}}$ inverse transform is $e^{-\frac{t}{\tau}}$. Hence, the inverse transform of Eq. 2.3 is:

**Eq. 2.4:** $V_o(t) = \frac{1V_s}{\tau} \cdot e^{-\frac{t}{\tau}}$,

whose waveform is shown in Figure 22. Figure 22 shows that the output voltage is 0 for $t<1ms$. However, for $t=1ms^*$ the output voltage is 1KV (i.e. $1Vs/\tau=1KV$). This means the output voltage is a discontinuous function despite the fact it is a voltage across a capacitor. How is this possible? We know, in fact, the voltage across a capacitor is a continuous function of time.
Figure 22: RC pulse response when R=1KΩ and C=1µF. The actual input voltage was a Dirac pulse delayed in time, i.e. δ(t-1ms). Note that τ=1ms and the transient lasts for 5ms (from 1 to 6ms) as expected.

The reason is that in real life there are not unbounded current and/or voltage values, hence the voltage across a capacitor as well as the current through an inductor are continuous functions of time. So we can safely use these continuity rules in all real circuits to find their initial voltage and current conditions. However, the Dirac pulse is not a real input voltage but it is simply an ideal waveform that is not present in real life. And, in fact, its value is +∞ for t=0, so the rule of voltage continuity across a capacitor doesn’t hold because both current and voltage values are not bounded at t=0.

So, why do we have to study the Dirac response of a system? The reasons are at least two. First of all, the actual pulse response of a system is approximately the same as the Dirac one scaled down by a suitable scale factor given by 1Vs/A, where A is the area below the actual pulse waveform. Secondly, the Laplace transform of the pulse response of a linear system is the transfer function of the system. So we may derive useful information about the transfer function of a system by studying its pulse response [15].

How is a Dirac pulse approximated in the laboratory? We may think of a pulse with a huge amplitude and a very small width, at least compared with the time constant of the system, as in the case shown in Figure 23. It is a trapezoidal pulse with amplitude 10KV and width 30µs, i.e. a factor 33 below the time constant of the system τ=1ms (a factor of 10 lower would already be sufficient to consider it a good approximation of a Dirac pulse). Figure 23 shows the output voltage waveform of the RC circuit that can be very well approximated with the formula:

Eq. 2.5: \[ V_o(t) = \frac{A}{\tau} e^{-\frac{t}{\tau}}, \]

where A=(10µs+30µs)*10KV/2=200mVs is the area below the input pulse voltage waveform.
Figure 23: Simulation of an RC pulse response when $R=1\,\text{K}\Omega$ and $C=1\,\mu\text{F}$. The first waveform is the bounded input voltage pulse while the second one is the system output voltage. Note the two x-axes of the plots have not the same scale.

Eq. 2.5 is a very good approximation for any kind of pulse (not only a trapezoidal one) whose width is significantly lower than the time constant of the system. It is sufficient to calculate the area $A$ below the pulse waveform and we are capable to determine the RC pulse response. Does this mean that the voltage across a capacitor is not continuous even when the input voltage and current values are bounded? Definitely not. In fact, Eq. 2.5 is only an approximation of the real pulse response of the system when we look at it in the ms (i.e. $\tau$) time scale. Figure 24 shows the initial transient of the RC output voltage in the $\mu$s (i.e. pulse width) time scale: there is no discontinuity in the pulse response. The output voltage rises from its initial 0V value to its maximum value during the pulse width period (in a very short time): the law of continuity holds because the input pulse amplitude is huge but bounded.

Figure 24: RC pulse response waveform of Figure 23 during the first 50$\mu$s of the transient. There is no voltage discontinuity across the capacitor because the input pulse is bounded. Only in a ms time scale it seems $V(\text{vo})$ has a discontinuity from 0V to 200mV.

To tell the truth completely, however, there is some discontinuity... Do you remember the ESR of the capacitor? The bounded but huge current at the beginning of the transient flows through the
capacitor ESR and produces a significant voltage drop across it whose value rises as fast as the input voltage. So, if the input voltage is discontinuous, the output voltage is discontinuous too, because of the voltage drop across the ESR. This should be taken accurately into account when analyzing the pulse response of a system. In conclusion, the pulse response of a system provides useful information about its transfer function property. In particular, the analysis of the pulse response with a spectrum analyzer gives a good approximation of the transfer function frequency dependence of the system. Since the transfer function frequency dependence behavior provides significant insights about the zeros and poles positions of the system, the stability properties of the system can be derived [16].

**Applications of Laplace transforms and transient linear systems response:** The main application of Laplace transforms is in the field of stability of analog automated control systems, with particular emphasis on analog feedback control systems [10,11]. However, this analysis is beyond the scope of this module and we won’t investigate it further.

A much simpler application of linear systems transient response is the determination of an unknown system parameter from the experimental observation of its transient response. This is a real problem we faced in the technology laboratory last year: we needed an inductor of about 30µH able to sustain up to 2A current to be used as part of a filter for a class D audio amplifier [17]. There were no available inductors in the lab and we needed to build it or to take it from an already existing PCB. In both cases we needed to measure it to be sure of its value. How did we do it?

We mounted an LR circuit with a known measured resistor value (a 1W 8Ω resistor - measured with a digital multimeter) and applied an input square voltage waveform from 0 to 10V to the circuit. The output voltage waveform measured with an oscilloscope was as in Figure 25:

![Figure 25: LR square wave response; R is known (8Ω) while L is the unknown parameter to be determined from the graph.](image)

From the waveform of Figure 25 we can evaluate the value of the time constant of the system τ with a graphical or an analytical approach.

1. **Analytical approach:** we know from the step transient response of the LR system that after a time constant τ the output voltage reaches 63.2% of its final value [18]. Since in our experiment the final voltage value is 10V, τ can be read from the graph as the time the output voltage takes to go from 0V to 6.32V (about 3.5µs).
2. Graphical approach: we trace the tangent to the waveform at the initial instant of the transient (t=0). The time the tangent takes to reach the final voltage value (i.e. 10V) is τ.

Both methods hopefully provide the same result, τ=3.5µs, as shown in Figure 25. Since τ=L/R we get L=τ*R=3.5µs*8Ω=28µH. The inaccuracy of the oscilloscope limits the precision of this technique to approximately a 10% error.

Linear systems are not only electrical circuits with capacitors, inductors and resistors. Many mechanical, hydraulic and thermal systems behave as linear systems and, in fact, they can be described and analyzed with the same techniques we have used for electrical linear systems, i.e. the finite difference method or the Laplace transform method. Moreover, there is a duality between all kinds of linear systems that allows us to describe any kind of linear system in the domain we prefer: that is to say, an electronic technician can describe a hydraulic system with an equivalent electronic circuit as well as a hydraulic technician can describe an electronic circuit with an equivalent hydraulic system [11]. Let’s see a practical example: consider the hydraulic system of Figure 26 a1) made of two tanks connected by a pipe closed with a valve (or a tap). Initially, the valve is closed and the two tanks are not communicating. The first tank is full with a certain volume of water Wᵢ, which makes a certain pressure Pᵢ on the surface of the tank. The second tank is empty (the volume of water is 0). At instant t=0 the valve is opened and the water starts to flow from the first tank to the second one. At the end of the transient the pressure of the water in the two tanks will be the same, Pᵓ, as shown in Figure 26 a2). What’s Pᵓ value (and hence Wᵢ₁, and Wᵢ₂ values) and what’s the transient behavior of the water flow in the two tanks?

![Figure 26: Figures a1) and a2) show the schematic of a simple hydraulic system made of two tanks connected by a pipe that can be closed with a valve. Figure b1) and b2) show the equivalent electric circuit, where voltage across capacitors corresponds to the water pressure on the tanks surfaces while the resistance current corresponds to the flow of water in the pipe.](image-url)
Figure 26 b1) - the voltage across the capacitor is equivalent to the pressure of the water on the tank surface. The pipe is equivalent to a resistance \( R \) and the valve is a switch initially open that is closed at instant \( t=0 \) (the current flowing through the resistance is equivalent to the rate of water flow in the pipe). So the transient behavior of the hydraulic system is the same as the one of the electronic circuit of Figure 26 b), that can be studied with the Laplace transform method.

The equivalent Laplace transformed circuit of Figure 26 b1) is shown in Figure 27, where the initial non-zero voltage across \( C_1 \) is taken into account in the voltage generator \( V_i/s \).

\[ \frac{1}{sC_1} \]
\[ R \]
\[ \frac{1}{sC_2} \]
\[ V_i/s \]
\[ + \]
\[ - \]
\[ V_{O1}(s) \]
\[ V_{O2}(s) \]

**Figure 27:** Equivalent Laplace transformed circuit of the circuit in Figure 26 b1). Note the initial voltage \( V_i \) across capacitor \( C_1 \) corresponds to the \( V_i/s \) voltage generator in the circuit. \( V_{O1}(s) \) (the voltage across capacitor \( C_1 \) in the Laplace domain) is therefore given by: \( V_i/s-I(s)/(s*C_1) \).

All circuits components are in series, hence circuit current can be easily calculated:

\[
I(s) = \frac{\frac{V_i}{s}}{\frac{1}{s*C_1} + R + \frac{1}{s*C_2}} = \frac{V_i}{1 + \frac{s*C_1}{C_1 + C_2}} \cdot \frac{C_1 \cdot C_2}{C_1 + C_2} = C_s \cdot V_i \cdot \frac{1}{1 + \frac{s*\tau_s}{R}} = \frac{V_i}{s} \cdot \frac{1}{s + \frac{1}{\tau_s}},
\]

where \( C_s = \frac{C_1 \cdot C_2}{C_1 + C_2} \) is the equivalent series capacitor, and \( \tau_s = R \cdot C_s \) is the time constant of the system. By using Eq. 2.6 we can easily calculate the expression of \( V_{O2}(s) \) and \( V_{O1}(s) \):

\[
V_{O2}(s) = \frac{1}{s*C_2} \cdot I(s) = \frac{V_i}{R \cdot C_2} \cdot \frac{1}{s \cdot (s + \frac{1}{\tau_s})}; V_{O1}(s) = \frac{V_i}{s} - \frac{1}{s \cdot C_1} \cdot I(s) = \frac{V_i}{s} - \frac{V_i}{R \cdot C_1} \cdot \frac{1}{s \cdot (s + \frac{1}{\tau_s})},
\]

From the Laplace transform table of Appendix A we get that \( \frac{1}{s \cdot \tau_s} \cdot (1 - e^{-\frac{t}{\tau_s}}) \) is the inverse transform of \( \frac{1}{s \cdot (s + \frac{1}{\tau_s})} \), while the inverse transform of \( \frac{V_i}{s} \) is \( V_i \). Hence, the inverse transforms of Eq. 2.7 are:

\[
V_{O2}(t) = \frac{V_i}{R \cdot C_2} \cdot \tau_s \cdot (1 - e^{-\frac{t}{\tau_s}}) = V_i \cdot \frac{C_1}{C_1 + C_2} \cdot (1 - e^{-\frac{t}{\tau_s}})
\]

**Eq. 2.8:**

\[
V_{O1}(s) = V_i - V_i \cdot \frac{C_2}{C_1 + C_2} \cdot (1 - e^{-\frac{t}{\tau_s}}) = V_i \cdot \frac{C_1}{C_1 + C_2} + V_i \cdot \frac{C_2}{C_1 + C_2} \cdot e^{-\frac{t}{\tau_s}}
\]
whose behavior is shown in Figure 28 for a given set of parameters values.

![Figure 28](image)

Figure 28: Evolution in time of the voltages across the two capacitors of the circuit of Figure 27 for the following values of the parameters: $V_i=10\text{V}$, $C_1=10\text{nF}$, $C_2=7.5\text{nF}$ and $R=1\text{K}\Omega$. The time constant of the circuit is about $4.3\mu\text{s}$.

Figure 28 shows the final voltage across the two capacitors is the same \( \left( V_i \cdot \frac{C_1}{C_1 + C_2} \right) \) and corresponds to a given final value of pressure of the water on the two tanks surfaces. In particular, the waveforms of Figure 28 correspond to the evolution in time of the pressure of water on the surfaces of the two tanks, from which the evolution in time of the volume of water in the two tanks can be derived.

In conclusion, all linear systems can be represented by an equivalent linear electronic circuit and vice-versa [11]. Therefore the Laplace transform method is a general way to determine the evolution in time of any kind of linear system (mechanical, hydraulic, thermal) present in nature.

We’ll conclude this Section by providing a user defined C library named `tranlib.cpp` in Appendix B which contains a set of functions that describe the pulse, step and ramp output voltage responses of the set of first and second order linear system shown in Table 1:

<table>
<thead>
<tr>
<th>Circuit</th>
<th>Dirac pulse: $\delta(t)$</th>
<th>Step input: $V_f\ast u(t)$</th>
<th>Ramp input: $P\ast t\ast u(t)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>RC</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>CR</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LR</td>
<td>X</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>RL</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>LC</td>
<td>X</td>
<td></td>
<td>X</td>
</tr>
<tr>
<td>CL</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RCL series</td>
<td>X</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Each X in the Table corresponds to a function in the C library `tranlib.cpp` provided in Appendix B and to a row of the Laplace transform table in Appendix A. $V_f$ is the step final voltage value (in V), while $P$ is the slope of the input ramp (in V/s). The functions in the library can be easily identified by their name, that is a combination of the circuit name and the input waveform.
In the question and exercise paragraph you’ll be asked to write C programs that use this library. Enjoy your work!

**Some odd results on linear systems – questions and exercises**

1) Draw a plot of the Dirac pulse waveform. Can we generate a Dirac pulse in the lab?

2) Can I use the finite difference method to find the Dirac pulse transient response of an RC system? And the RC transient response to a bounded pulse of finite width $t_0$?

3) Implement an LR linear system in the lab and apply a bounded pulse of width $t_0$ to it. What output waveform do you expect? Is there current continuity in the inductor?

4) Explain the reason why the step response of a CL system features damping oscillations. What would the step response of an ideal CL system be?

5) Write a C program that calculates the step response of an RCL series system. The program reads the values of the circuit components (R, C and L) and the final value of the step input voltage (Vf). The output of the program should be a text file that contains the waveforms of $V_R(t)$, $V_C(t)$ and $V_L(t)$ (the voltages across the resistor, capacitor and inductor, respectively). Finally, the text file should be imported in Excel to draw the corresponding waveforms.

6) Write a C program that asks the user what kind of linear circuit (system) and input stimulus he wants to solve among the options shown in Table 1. The program output is a text file containing the waveform of the desired transient response of the selected system.

**SECTION 3: SIMULATION OF FINITE STATE MACHINES BY C PROGRAMMING**

**Introduction:** Linear systems are a basic component of many analog control systems and the study of their transient response provides us with useful information about the stability of the control system, which is one of the most important characteristics when dealing with control issues [10, 11].

However, many control systems are digital, i.e. their inputs and outputs are 0V (false) or 5V (true), and they cannot be studied by means of transient response analysis. Combinational circuits have digital inputs and outputs too, but they are not digital control systems. Hence, a digital control system has something more than digital inputs and outputs. In fact, the output of a digital control system is not only a function of its digital inputs but it depends on the internal state of the system too. Let’s make a practical common life example: the remote control of your house automatic gate. The opening/closing of the gate is controlled by a remote control device with only one button. When you push the button what happens? It depends on the state of the system: if the gate is in its initial state (closed) when you push the button the gate starts opening. If, on the contrary, the gate is closing and you push the button, the gate stops closing and after a certain time it starts opening. So
the same input (you push the button) translates into two different actions depending on the state of the system.

In general, a digital control system whose outputs at instant \( t' \) depend not only on the value of the inputs at the same instant but on the past history of the system too (on the system memory) is called finite state machine when it is possible to identify: a finite number of digital inputs; a finite number of digital outputs; a finite number of internal states, among which it is possible to identify an initial and a final state \([19,20]\). The inputs of a finite state machine are usually indicated with the variable \( X \) \((X=[x_n:x_1]\) when there are \( n \) digital inputs), the outputs with the variable \( Y \) \((Y=[y_m:y_1]\) when there are \( m \) digital outputs) and the variables associated to the internal states of the system are indicated with \( Q \) \((Q=[q_k:q_1]\) when there are \( 2^k \) or less internal states – in fact, the number of internal state variables is the minimum integer number greater than or equal to \( \log_2 k \), where \( k \) is the total number of internal states). So, in formulas, a finite state machine is described by the relationship: \( Y=f(X,Q) \), that can be read as follows: the outputs \( Y \) of a finite state machine are a function \( f \) of the inputs \( X \) and of the internal states variables \( Q \).

State Graph representation of finite state machines: it is important to remember that the number \( 2^k \) of internal states \( (S_2;…;S_1) \) of a finite state machine corresponds to only \( \log_2(2^k) = k \) internal state variables \([q_k:q_1]\). The first difficulty to design a finite state machine is to start from its written/verbal description and represent it by means of the State Graph. First of all we may have two different kinds of State Graphs: the Moore State Graph, where each state \( S_i \) is associated to a particular value of the outputs \( Y' \), and the Mealy State Graph, where each transition of state from \( S_i \) to \( S_j \) is associated with a particular value of the outputs \( Y \) (or with an action affecting the outputs). The two descriptions are usually equivalent. However, in some cases, the Mealy description is more efficient (it provides a lower number of internal states and, hence, of internal state variables), especially in those systems where an action is associated to each state transition.

Since it is difficult to derive all the possible states of a finite state machine from its verbal description, we’ll start with a simple example: the remote control of the turn on – turn off of a lamp. We have a remote control device with a button; when we push the button we want the lamp to change its state: if it was on we want it to turn off; if it was off we want it to turn on. This simple finite state machine has one digital input \( x_1 \) describing the state of the button: while the button is pushed \( x_1=1 \) (true, or 5V), when the button is released \( x_1=0 \) (false or 0V). A simple electronic circuit that is able to convert the pressure/release of a button to a digital signal 5 or 0V is shown in Figure 29. It contains an RC circuit simply to prevent bouncing of the input voltage that would result in undesired input level switching each time the button is pushed/released \([21]\).

The remote control device does not contain simply the circuit of Figure 29. It also contains a wireless transmitter (TX) circuit, based on infrared (IR) or radio frequency (RF) technology, to transfer the digital signal \( x_1 \) to the lamp driving circuit \([22]\). The actual lamp driving circuit is connected to the lamp by cables. It contains a wireless receiver (RX) circuit based on the same technology as the transmitter of the remote control device. The receiver circuit output will be a digital signal whose value is the same as \( x_1 \) and is used to drive our simple finite state machine, also contained in the lamp driving circuit.
The finite state machine has one digital output $y_1$ too: $y_1=0$ (false or 0V) corresponds to lamp OFF state, $y_1=1$ (true or 5V) corresponds to lamp ON state. How is it possible to turn-on/off a lamp with a digital signal? Of course, we cannot directly turn-on/off a lamp with a digital signal. In fact, the lamp driving circuit contains an actuator too: an actuator has a high input impedance (and hence it can be driven by a low drive capability signal such as $y_1$), and at the same time it is able to deliver the sufficient power to drive high power loads such as lamps, motors, etc… Figure 30 shows all the blocks of the circuit to turn-on/off the lamp when $y_1=1/0$: the receiver, the finite state machine and the relay that connects/disconnects the lamp to the 220V depending on finite state machine output $y_1$.

From the analysis above it is clear that we are simply interested in the finite state machine design which involves only digital signals. However, a complete digital control system includes sensors, actuators and telecommunication devices that may work with mixed signals (analog and digital signals...).
electronic signals). In particular, sensors are needed to transform physical inputs/quantities into electrical signals (the circuit of Figure 29 is a sensor, because the pressure on the button is transformed into a digital signal) [23]. Telecommunication circuits are needed to transfer the information (digital or analog signals) among circuits located in different space positions that cannot be simply connected by means of a simple wire [24]. Actuators are needed to drive the load, because the finite state machine outputs feature a poor drive capability [25].

The core of the whole control system is however the finite state machine that, in our case, has only $x_1$ input and $y_1$ output. The behavior of the finite state machine can be described by the State Graph or the time diagram. The time diagram shows how the finite state machine outputs evolve in time as a function of the inputs. Figure 31 is the time diagram of our finite state machine:

Each time the button is pushed ($x_1=1$) the value of $y_1$ changes and the lamp state changes accordingly. The initial state $S_1$ of the system is with $x_1=0$, $y_1=0$. When $x_1$ becomes 1 the system goes from state $S_1$ to state $S_2$ with $y_1=1$. As soon as the button is released the system goes into state $S_3$, with $x_1=0$, and $y_1=1$. At the next button pressure the system goes from state $S_3$ to state $S_4$, with $x_1=1$, and $y_1=0$. When the button is released the system returns to state $S_1$, the initial one, with $x_1=0$, $y_1=0$.

Figure 32 shows the Moore and the Mealy State Graph of the system:

As it can be seen, there is no difference between the number of states in the Moore and Mealy description of our simple finite state machine. Note the following rules that always apply to Mealy and Moore State Graphs of asynchronous finite state machines:
• Each state input arrow corresponds to an auto-retention arrow (only for asynchronous FSM’s).

• The total number of each state output arrows is equal to \(2^n\), where \(n\) is the number of inputs (in our system there is only \(x_1\) input, i.e. 1 input, hence \(2^1=2\) output arrows for each state, one of which is an auto-retention arrow).

**Implementation of finite state machines:** once we have the State Graph we can proceed with the design of the finite state machine. There are two possible ways to design the finite state machine:

- HW (Hardware) implementation with discrete digital components (logic gates and FFs);
- SW (Software) implementation with a microcontroller.

The HW implementation is the realization of a Printed Circuit Board (PCB) with discrete digital components, mainly logic gates and Flip-Flops, combined to form a digital circuit that behaves as described in State Graph [26]. The advantage of this solution is the speed of the system response to inputs variations; the disadvantage is its complexity and the difficulty to implement modifications and improvements: once the PCB is done, if you want to add some functions it is nearly always necessary to re-build another PCB.

The SW implementation is the realization of a Printed Circuit Board (PCB) with only a microcontroller (for instance PIC16F84A by Microchip technologies) that is programmed in order to behave as described in the State Graph [27]. The advantage of this solution is its flexibility: we may change the State Graph that it is sufficient to modify the program (the SW) stored within the microcontroller; we may add some features in a second moment simply by adding some code lines in the microcontroller program. The disadvantage is that the system response to input variations is not as fast as the HW implementation solution. However, the speed of microcontroller systems is usually very fast compared with typical control requirements, so this is not a real issue. Therefore, most finite state machines are implemented with an SW programmed microcontroller. Control systems microcontroller based or microprocessor based are nowadays largely spread in a variety of industrial applications and they represent the most common kind of control system.

**HW Implementation:** Let’s proceed with the HW implementation of our simple finite state machine considering the Moore’s State Graph of Figure 32. Since we have 4 different internal states, we need \(\log_24=2\) state variables, \(q_2\) and \(q_1\). Each possible combination of state variables corresponds to an internal state \(S_i\) and, hence, a value of the output \(y_1\) according to Table 2a). Table 2a) can be translated in the equivalent Karnaugh map shown in Table 2b), from which it is immediate to determine the relationship between \(y_1\) and the state variables: \(y_1=q_1\). \(y_1\) is independent of \(q_2\) value.

The next step is to determine the sequential circuit that generates \(q_1\) and \(q_2\) (the actual memory of the finite state machine). The simplest choice (but not always the most efficient one) is to use one DFF (D type Flip Flop) for each state variable. We need two DFF for our finite state machine. Each DFF input, \(d_1\) and \(d_2\), will depend on the state variables \(q_1, q_2\) and the input \(x_1\).
Table 2 a) Correspondence between internal state variables and output value (this is called the Table of the outputs). b) Karnaugh map relative to $y_1$ output.

<table>
<thead>
<tr>
<th>State</th>
<th>$q_2$</th>
<th>$q_1$</th>
<th>$y_1$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$S_1$</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>$S_2$</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$S_3$</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$S_4$</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 3 a) Transition table providing the future (at the $N^{th}+1$ clock rising edge) internal state as a function of the present state (between $N^{th}$ and $N^{th}+1$ clock rising edge) and the input value $x_1$ (at the $N^{th}+1$ clock rising edge). b) Karnaugh map for $d_1$ input of the $q_1$ DFF as derived from the two highlighted columns of the Transition table.

<table>
<thead>
<tr>
<th>$q_2^N$</th>
<th>$q_1^N$</th>
<th>$x_1=0$</th>
<th>$x_1=1$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 3 a) Transition table providing the future (at the $N^{th}+1$ clock rising edge) internal state as a function of the present state (between $N^{th}$ and $N^{th}+1$ clock rising edge) and the input value $x_1$ (at the $N^{th}+1$ clock rising edge). b) Karnaugh map for $d_1$ input of the $q_1$ DFF as derived from the two highlighted columns of the Transition table.

The use of DFF implies a clock signal at whose rising edge $d_1$ and $d_2$ are sampled. Actually, all the digital signals of the finite state machine ($x_1$ and $q_1$, $q_2$) will be evaluated (sampled) at the clock rising edge: as a consequence the output signal $y_1$ is updated at the clock rising edge too. In particular, $d_1$ at any instant between the $N^{th}$ and the $N^{th}+1$ clock rising edge is the value $q_1$ should take at the $N^{th}+1$ rising edge ($d_1=q_1^{N+1}$). The same occurs for $d_2$. Table 3a) reports the transitions table of our finite state machine, i.e. the dependence of the next state (N+1) from the previous one (N) and the value of $x_1$ during the $N^{th}+1$ clock rising edge. Since $d_1=q_1^{N+1}$, the two columns highlighted in Table 3a) represent the Karnaugh map for $d_1$, as shown in Table 3b). From the Karnaugh map the logic dependence of $d_1$ on $q_1$ ($q_1^N$), $q_2$ ($q_2^N$), and $x_1$ is $d_1=(x_1 \& \& (!q_2)) \lor ((!x_1) \& \& q_1))$. A similar Karnaugh map for $d_2$ provides $d_2=(x_1 \& \& q_2) \lor ((!x_1) \& \& q_1))$ – note we are using C programming conventions for logic operators. The clock frequency choice determines the sampling period of the system digital signals: a 1KHz clock frequency corresponds to a 1ms sampling period. This means the pressure on the button should last for at least 1ms to be detected correctly. The duration of the button pressure is actually limited by the debounce circuit of Figure

---

4 $|$ is the logical OR, $\&\&$ is the logical AND and $\&\&$ is the logical OR.
29, whose time constant is approximately $15K \cdot 0.1 \mu s = 1.5 ms$ corresponding to a total transient time of $5*\tau = 7.5 ms$. So the pressure on the button should last at least $10 ms$ to allow the full discharge from $5V$ to $0V$ of the digital signal $x_1$ and its correct detection. This is not a real issue for common switches of actual remote control systems.

In conclusion, our finite state machine is described by the following logic functions:

$$\begin{align*}
    y_1 &= q_1 \\
    d_1 &= (x_1 \land \lnot (q_2)) \lor ((\lnot x_1) \land q_1) \\
    d_2 &= (x_1 \land q_2) \lor ((\lnot x_1) \land q_1)
\end{align*}$$

The logic formulas of Eq. 3.1 correspond to the digital sequential circuit of Figure 33, that is the HW implementation of our finite state machine:

![Figure 33](image)

**Figure 33** HW implementation of the finite state machine with discrete digital components and DFF’s. The block “finite state machine” of Figure 30 contains this circuit.

As it can be seen, the synthesis process of the finite state machine requires many steps. Furthermore, the final circuit cannot be easily modified nor upgraded to include further functionalities. Another problem is that discrete digital components are nowadays obsolete devices: a single quad-nand gate occupies an area equal to that of an 8 bit microcontroller and the PCB containing the circuit of Figure 33 has a very huge area. For all these reasons actual digital control systems are not implemented with digital discrete components but are microcontroller based [28]. A single 8 bit microcontroller (the smallest among the microcontroller device family) integrates in its small area an equivalent number of logic gates as high as 10000 or 100000 depending on its fabrication technology [29]. Furthermore, it is programmable, that means its behavior can be changed simply by changing the program stored in its internal non-volatile memory. So, it is easy to add further functionalities or to modify a finite state machine microcontroller based even after its PCB implementation. However, the concepts behind the design of finite state machines are still used in semiconductor industries in those cases where small custom sequential digital blocks are needed within a more complex mixed-signal integrated circuit (IC). Complex digital integrated circuits are instead designed by means of specifically developed programming languages, among which the most widely used are VHDL (VHSIC Hardware Description Language – where VHSIC stands for Very High Speed Integrated Circuits) and Verilog [30,31]. VHDL and Verilog are very similar to the C programming language and today they represent the most widely used tools to design complex...
digital integrated circuits (digital IC’s) such as DSP’s (Digital Signal Processors), microcontrollers and microprocessors. The main difference between the HW description languages and conventional programming languages as C is they are concurrent and not sequential, i.e. the instructions are not executed one after the other according to the order in which they are written in the programming code.

**Sometimes Mealy is more efficient:** the case of our finite state machine is one where the Mealy State Graph can be simplified. In fact, the behavior of the finite state machine can be summarized as follows: the output \( y_1 \) toggles at each rising edge of the input \( x_1 \). To toggle means to change state: when \( y_1 \) is 0 it becomes 1 and vice-versa. The State Graph of Figure 32b) can therefore be simplified as shown in Figure 34a):

\[
\begin{align*}
S_1 & \quad x_1 = 0/1 \quad y_1 = y_1/\neg y_1 \\
S_2 & \quad x_1 = 0/1 \quad y_1 = y_1/\neg y_1
\end{align*}
\]

**Figure 34** a) Simplified Mealy State Graph of the finite state machine, with only two internal states. Note that each transition of state is not associated to an output value but to an action (in this case the toggling of the output variable \( y_1 = \neg y_1 \) or no action at all indicated by - ). b) Equivalent circuit of the State Graph in a), with a single TFF.

A State Graph with only two states is implemented by a sequential circuit with only one DFF. In particular our system is equivalent to a single TFF (T kind Flip Flop), as shown in Figure 34b): each \( x_1 \) rising edge, the output \( y_1 \) of the TFF toggles since the T input of the FF is stacked at 1. Note that the circuit of Figure 34b) is not sequential (there is no clock signal) but it is asynchronous, with the advantage that each \( x_1 \) rising edge affects the output immediately without the need to wait for any clock rising edge. In general, Mealy State Graphs are convenient for those digital control systems where each transition of state can be associated with an action rather than a particular value of the outputs \( Y \).

**SW Implementation:** the SW implementation of the finite state machine is the microcontroller based solution. We named it SW implementation because the behavior of the finite state machine is univocally defined by the program (a program is a software) stored in the microcontroller non-volatile memory. The microcontroller chosen to implement the finite state machine is the PIC16F84A (also called simply PIC), an 8 bit microcontroller by Microchip Inc. packaged in an 18 pins DIP (Dual In line Package) [29]. Of course, a PCB is still required where to solder the PIC and the basic external discrete components (mainly capacitors and resistors) the microcontroller needs to work correctly, as shown in Figure 35. For a given PCB, many finite state machines can be designed by simply modifying the program code (SW) stored in the PIC’s EEPROM (Electrically Erasable Programmable Read Only Memory). Once the PCB is fixed, however, not all finite state machines State Graphs can be implemented by simply modifying the SW.

For our simple finite state machine we may only use two pins of PIC’s PORTB, namely RB4 (COL0 signal in Figure 35) as \( x_1 \) input and RB5 (COL1 signal in Figure 35) as \( y_1 \) output. All other PIC’s I/O (Input/Output) are not used by our finite state machine. This means RB4 should be
connected by a wire to the output of the RX block of Figure 30, that provides $x_1$ logic signal. On the other hand, RB5 is connected with a wire to the base of the BJT transistor of Figure 30 ($y_1$ signal). In practice, the PIC is in place of the finite state machine block of Figure 30.

**Figure 35** Example of PCB required for an SW implementation of a finite state machine. PIC16F84A is the 8 bit microcontroller by Microchip Inc. we use in the laboratory. The actual connections of the PIC I/O (Input/Outputs) ports depend on the application. However, for a given set of connections, many different applications can be addressed by simply modifying the code (SW) stored in PIC’s non-volatile memory.

After PCB connections are defined, the last task is to write the SW into the PIC’s EEPROM. Almost all microcontrollers can be programmed in two ways:

- with an Assembly language program (low level, very close to the HW level);
- with a C language program (high level programming).

The Assembly language is very close to the machine language of the microcontroller [32]. Each microcontroller has its own set of assembly instructions (for instance PIC16F84A features approximately 35 different assembly instructions) that are translated by an Assembler program into the machine code that is directly executed by the microcontroller CPU (Central Processing Unit). It is not straightforward to write assembly code, even if the best code efficiency can only be reached by low-level programming.

All microcontrollers support high level programming to simplify the job of SW developers, and, in particular, C programming is the most widely spread among the different microcontrollers manufacturers [33]. Each microcontroller manufacturer provides an environment freely where to develop the C source code, compile it and then program the resulting machine code into the microcontroller’s EEPROM or Flash memory. For instance, in Microchip MPLAB environment there is also a simulator to analyze the behavior of the microcontroller before operating it on the field, allowing for significant debugging time saving.

To design a microcontroller based finite state machine it is sufficient to write a C code that implements its State Graph. Considering the State Graph of Figure 32a) we have four different states $S_i$ with $i$ from 1 to 4. So we may define four different bool (Boolean type) variables and call
them S1, S2, etc… or define two internal state variables q1 and q2 whose combination of values identifies one of the four states. We’ll choose the four state variables: the program will be less efficient but more readable. When the device is in one State, for instance S1, then we have to:

- assign the output y1 the value corresponding to the State the system is;
- check for input values able to cause state variations and change the state accordingly.

Here follows the C code corresponding to the State Graph of Figure 32a):

```c
// program defined input data dependent on PCB connections
#define x1   RB4   // input variable is the digital value at pin RB4
#define y1   RB5   // output variable is the digital value at pin RB5

// program variables definition
bool S1=1, S2=0, S3=0, S4=0; // initial state is S1

int main (){ // initialization
    y1=0; // output value corresponding to initial state S1 is 0

    for(;;){ // finite state machine working cycle
        if (S1){
            y1=0; // update output value with the value corresponding to the actual state
            if (x1){
                S1=0, S2=1; // transition from state S1 to S2 when x1=1
            }
        } else if (S2){
            y1=1; // update output value with the value corresponding to actual state
            if (!x1){
                S2=0, S3=1; // transition from state S2 to S3 when x1=0
            }
        } else if (S3){
            y1=1; // update output value with the value corresponding to actual state
            if (x1){
                S3=0, S4=1; // transition from state S3 to S4 when x1=1
            }
        } else if (S4){
            y1=0; // update output value with the value corresponding to actual state
            if (!x1){
                S4=0, S1=1; // from state S4 to S1 when x1=0
            }
        }
        // close for cycle
    } // close main

    Figure 36: code of the ANSI-C program that implements the State Graph of Figure 32a).

As it can be seen the C code is very simple and easy to write. There is an infinite for loop because the finite state machine should always work. During the loop the state of the system is continually checked and the outputs updated accordingly to the state the system is. Depending on system state the input is checked too in order to verify for possible state transitions. Most importantly, the digital control system will effectively work only if its inputs and outputs are correctly connected. Suppose for instance pin RB4 has not been connected to RX block digital output x₁ and it is instead connected to the VDD supply (5V). This means that the program x1 variable value is always 1 and hence the system immediately passes from S1 to S2 and then remains forever in state S2, independently on RX x₁ digital value. On the other hand, if the BJT base is not connected to RB5 but to another PIC pin whose value is for instance stacked at 0, the BJT and hence the relay and the lamp will always be off. So the correspondence between the I/O defined in the C program and the connections in the PCB is extremely important.
What about the Mealy State Graph of Figure 34a)? When the device is in one State, for instance S1, we have to check first for input values able to cause state variations, perform the corresponding action and then change the state accordingly.

Here follows the C code corresponding to the State Graph of Figure 34a):

```c
// program defined input data dependent on PCB connections
#define x1 RB4  // input variable is the digital value at pin RB4
#define y1 RB5  // output variable is the digital value at pin RB5

// program variables definition
bool S1=1, S2=0;  // initial state is S1

int main (){  
    // initialization
    y1=0; // initial output value is 0
    for(;;){  // finite state machine working cycle
        if (S1){
            if (x1){
                y1=!y1;  // perform the action – toggle y1
                S1=0, S2=1; // transition from state S1 to S2 when x1=1
            }
        }
        if (S2){
            if (!x1){ // perform no action
                S2=0, S1=1; // transition from state S2 to S1 when x1=0
            }
        }
    }  // close for cycle
}  // close main
```

**Figure 37:** code of the ANSI-C program that implements the State Graph of Figure 34a).

The program of Figure 37 can be further optimized as follows:

```c
// program defined input data dependent on PCB connections
#define x1 RB4  // input variable is the digital value at pin RB4
#define y1 RB5  // output variable is the digital value at pin RB5

// program variables definition
bool q=1;  // q=1 is S1, q=0 is S2

int main (){  
    // initialization
    y1=0; // initial output value is 0
    for(;;){  // finite state machine working cycle
        if (q && x1){
            y1=!y1;  // perform the action – toggle y1
            q=0; // transition of state when x1=1
        }
        if (!x1) q=1; // return to initial state
    }  // close for cycle
}  // close main
```

**Figure 38:** optimized code of the ANSI-C program of Figure 37.

With less than 10 C instructions we have designed our finite state machine: that’s simple!

Note that the only part of the program of Figure 36-38 that is HW dependent is the definition of the input and output variables x1 and y1. The algorithm and the main program is however unaffected by the I/O variables definition. Hence, we can easily simulate any kind of finite state machine independently on its actual HW implementation by means of a PC. The inputs will be one or more keyboards characters, the outputs will be some phrases written on the PC screen. For instance, the program of Figure 38 can be simulated on a PC as follows:

```c
#include <stdio.h>
#include <conio.h>

// program variables definition
bool q=1, y1=0;  // q=1 is S1, q=0 is S2; y1=0 is LAMP OFF, =1 is LAMP ON

int main (){  
    // initialization
    printf("The Lamp is OFF; press any key if you want to turn it on: ");
    getch( );

    while (q){  // finite state machine working cycle
        if (x1){
            y1=!y1; // perform the action – toggle y1
            q=0; // transition of state when x1=1
        }
        if (!x1) q=1; // return to initial state
    }
    // close while
}```
for(;;){  // finite state machine working cycle
    if (q && !_kbhit()){
        _getch();  // reset input buffer
        y1=!y1;  // perform the action - toggle y1
        if (y1) printf("The Lamp is ON\n");
        else printf("The Lamp is OFF; press any key if you want to turn it on: ");
        q=0;  // transition of state when x1=1
    }
    if (!_kbhit()) q=1;  // return to initial state
}  // close for cycle
}  // close main

Figure 39: PC simulation of the finite state machine of Figure 36-38.

_kbhit() function returns 1 value when any keyboard key is pressed (when there is at least one character in the input buffer). It returns 0 after the input buffer is void (after the _getch() function call). Unfortunately, this is true even if the pressure on the keyboard key is continuous (for the PC when you push a key for a long time it is as if you want to enter the character corresponding to that key many times – try it by opening a word processor: push the key ‘a’ and keep it pushed. On the sheet you’ll see aaaaaaaaaaa… and not a single a. So it is as if you push/pull the character ‘a’ a lot of times, while you simply pushed it once and kept it pushed). Hence, the program of Figure 39 will work as you expect only if the key pressure is single and fast. Apart from this inconvenience related to the _kbhit() function, we may use it to simulate a digital input and hence perform PC simulations of finite state machines, as asked in the next questions and exercise section. This is a useful exercise to prepare real design of microprocessor based finite state machines and, ultimately, digital control systems. Good work!

Finite state machines – questions and exercises

1) Explain all the optimization performed to pass from the C program of Figure 37 to the one of Figure 38.

2) With your PC simulate a finite state machine able to recognize one name made of four letters, such as for instance MARY. In practice the system should recognize a sequence of characters (for instance ‘M’, ‘A’, ‘R’ and ‘Y’) and then print Hello Mary! on the screen.

3) With your PC simulate a finite state machine able to control the automatic gate of your house.

4) With your PC simulate a finite state machine able to control the stop and go light of a car park. The car park can host a maximum of N cars. When a car enters the park a sensor provides the digital signal x1=1 in output; when no car is entering the park x1=0. When a car goes out another sensor provides the digital signal x2=1 in output; when no car goes out of the park x2=0. When the number of cars in the park is N the park light should be red, otherwise green. When green the number of vacancies should be displayed, too.
References

[20] http://www.youtube.com/watch?v=HyUK5RAJg1c
### Appendix A

Laplace Transform Table for first and second order electronic linear systems

<table>
<thead>
<tr>
<th>Waveforms</th>
<th>Inverse transform</th>
<th>Transform</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dirac pulse</td>
<td>δ(t)</td>
<td>1 (Volt\times\text{s})</td>
<td>Dirac pulse</td>
</tr>
<tr>
<td>Step voltage</td>
<td>V_f \cdot u(t)</td>
<td>\frac{V_f}{s}</td>
<td>Step voltage</td>
</tr>
<tr>
<td>Ramp voltage</td>
<td>P \cdot t \cdot u(t)</td>
<td>\frac{P}{s^2}</td>
<td>Ramp voltage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Time domain (variable t, measured in seconds)</th>
<th>Laplace domain (variable s, measured in 1/seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Volt</td>
<td>Invert</td>
</tr>
<tr>
<td>Volt</td>
<td>Invert</td>
</tr>
<tr>
<td>Volt</td>
<td>Invert</td>
</tr>
</tbody>
</table>

### Waveforms

- **Dirac pulse**
- **Step voltage**
- **Ramp voltage**
RC, LR Dirac response and CR, RL step response
\[ e^{-a \cdot t} \quad \frac{1}{s + a} \]

RC, LR step response and CR, RL ramp response
\[ \frac{1}{a} \cdot \left(1 - e^{-a \cdot t}\right) \quad \frac{1}{s \cdot (s + a)} \]

RC, LR ramp response
\[ \frac{1}{a^2} \cdot \left(a \cdot t - 1 + e^{-a \cdot t}\right) \quad \frac{1}{s^2 \cdot (s + a)} \]
<table>
<thead>
<tr>
<th>Graph</th>
<th>Equation</th>
<th>Laplace Transform</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image" alt="LC Dirac pulse response and CL ramp response" /></td>
<td>$\text{sen}(\omega \cdot t)$</td>
<td>$\frac{\omega}{s^2 + \omega^2}$</td>
</tr>
<tr>
<td><img src="image" alt="LC step response" /></td>
<td>$\frac{1}{\omega^2} \cdot (1 - \cos(\omega \cdot t))$</td>
<td>$\frac{1}{s \cdot (s^2 + \omega^2)}$</td>
</tr>
<tr>
<td><img src="image" alt="CL step response" /></td>
<td>$\cos(\omega \cdot t)$</td>
<td>$\frac{s}{s^2 + \omega^2}$</td>
</tr>
<tr>
<td>RLC series step response with $\zeta&gt;1$ (overdamped)</td>
<td>$V_R(t)$</td>
<td>$\frac{1}{b-a} \cdot \left( e^{-a \cdot t} - e^{-b \cdot t} \right)$</td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>RLC series step response with $\zeta=1$ (critically damped)</td>
<td>$V_R(t)$</td>
<td>$t \cdot e^{-a \cdot t}$</td>
</tr>
<tr>
<td>RLC series step response with $\zeta&lt;1$ (underdamped)</td>
<td>$V_R(t)$</td>
<td>$\frac{1}{\omega_n \sqrt{1 - \zeta^2}} \cdot \sin(\omega_n \cdot \sqrt{1 - \zeta^2} \cdot t)$</td>
</tr>
</tbody>
</table>
Appendix B

C code of the library tranlib.cpp

/* This library contains some functions that write on an output file with the
same name as the function the transient response of the RC, CR, LR, RL, LC, CL
and RCL series circuits to a stimulus among the Dirac pulse, step or ramp waveform.
Each function is named according to the kind of circuit and the stimulus applied.
Each waveform is made of Np discrete time points distant the same time interval named step one from the other. The total number
of points is sufficient to cover the whole transient or, in case of periodic response, at least two periods of the waveform. */

#include <math.h>
#include <stdio.h>

/* functions definitions - prototypes */
void RCdirac(float R, float C, int Np);
void LRdirac(float R, float L, int Np);
void LCdirac(float L, float C, int Np);
void RCstep(float R, float C, float Vf, int Np);
void LRstep(float R, float L, float Vf, int Np);
void CRstep(float R, float C, float Vf, int Np);
void RLstep(float R, float L, float Vf, int Np);
void LCstep(float L, float C, float Vf, int Np);
void CLstep(float L, float C, float Vf, int Np);
void RCLstep(float R, float L, float C, float Vf, int Np);
void RCramp(float R, float C, float P, int Np);
void LRramp(float R, float L, float P, int Np);
void CRramp(float R, float C, float P, int Np);
void RLramp(float R, float L, float P, int Np);
void CLRamp(float L, float C, float P, int Np);

/* functions body */
void RCdirac(float R, float C, int Np){ // Dirac response of RC circuit
    float tau,step;
    int i;
    FILE *fp;

    fp=fopen("RCdirac.txt","w");
    fprintf(fp,"\nDirac response of RC circuit\n\nt[s]\tVo(t)[V]\n\n");
    tau=R*C, step=5*tau/Np;
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g\t%g",i*step,exp(-i*step/tau)/tau);
    fclose(fp);
}

void LRdirac(float R, float L, int Np){ // Dirac response of LR circuit
    float tau,step;
    int i;
    FILE *fp;

    fp=fopen("LRdirac.txt","w");
    fprintf(fp,"\nDirac response of LR circuit\n\nt[s]\tVo(t)[V]\n\n");
    tau=L/R, step=5*tau/Np;
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g\t%g",i*step,exp(-i*step/tau)/tau);
    fclose(fp);
}

void LCdirac(float L, float C, int Np){ // Dirac response of LC circuit
    float wn,step;

    ...
int i;
FILE *fp;

fp=fopen("LCdirac.txt","w");
fprintf(fp,"Dirac response of LC circuit\n");
printf(fp,"int[i]tVo(t)[V]\n");
wn=1/sqrt(L*C), step=4*M_PI/(wn*Np);
for(i=0;i<Np;i++)
    fprintf(fp,"\n%g\t%g\n",i*step,wn*sin(wn*i*step));
fclose(fp);

void RCstep(float R, float C, float Vf, int Np){ /* step response of RC circuit
    float tau,step;
    int i;
    FILE *fp;

    fp=fopen("RCstep.txt","w");
    fprintf(fp,"RC step response; final step voltage = %fV\n",Vf);
    fprintf(fp,"int[i]tVo(t)[V]\n");
    tau=R*C, step=5*tau/Np;
    for(i=0;i<Np;i++)
        fprintf(fp,"\n%g\t%g\n",i*step,Vf*(1-exp(-i*step/tau)));
    fclose(fp);
}

void LRstep(float R, float L, float Vf, int Np){ /* step response of LR circuit
    float tau,step;
    int i;
    FILE *fp;

    fp=fopen("LRstep.txt","w");
    fprintf(fp,"LR step response; final step voltage = %fV\n",Vf);
    fprintf(fp,"int[i]tVo(t)[V]\n");
    tau=L/R, step=5*tau/Np;
    for(i=0;i<Np;i++)
        fprintf(fp,"\n%g\t%g\n",i*step,Vf*(1-exp(-i*step/tau)));
    fclose(fp);
}

void CRstep(float R, float C, float Vf, int Np){ /* step response of CR circuit
    float tau,step;
    int i;
    FILE *fp;

    fp=fopen("CRstep.txt","w");
    fprintf(fp,"CR step response of CR circuit; step final voltage = %fV\n",Vf);
    fprintf(fp,"int[i]tVo(t)[V]\n");
    tau=C*R, step=5*tau/Np;
    for(i=0;i<Np;i++)
        fprintf(fp,"\n%g\t%g\n",i*step,Vf*exp(-i*step/tau));
    fclose(fp);
}

void RLstep(float R, float L, float Vf, int Np){ /* step response of RL circuit
    float tau,step;
    int i;
    FILE *fp;

    fp=fopen("RLstep.txt","w");
    fprintf(fp,"RL step response of RL circuit; step final voltage = %fV\n",Vf);
    fprintf(fp,"int[i]tVo(t)[V]\n");
    tau=L/R, step=5*tau/Np;
    for(i=0;i<Np;i++)
        fprintf(fp,"\n%g\t%g\n",i*step,Vf*exp(-i*step/tau));
    fclose(fp);
}
void LCstep(float L, float C, float Vf, int Np) {
    // step response of LC circuit
    float wn, step;
    int i;
    FILE *fp;
    fp=fopen("LCstep.txt","w");
    fprintf(fp,"\nLC step response; step final voltage = %fV\n",Vf);
    fprintf(fp,"t[s]\tVo(t)[V]\n");
    wn=1/sqrt(L*C), step=4*M_PI/(wn*Np);
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g\t%g\n",i*step,Vf*(1-cos(wn*i*step)));
    fclose(fp);
}

void CLstep(float L, float C, float Vf, int Np) {
    // step response of CL circuit
    float wn, step;
    int i;
    FILE *fp;
    fp=fopen("CLstep.txt","w");
    fprintf(fp,"\nCL step response; step final voltage = %fV\n",Vf);
    fprintf(fp,"t[s]\tVo(t)[V]\n");
    wn=1/sqrt(L*C), step=4*M_PI/(wn*Np);
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g\t%g\n",i*step,Vf*cos(wn*i*step));
    fclose(fp);
}

void RCLstep(float R, float L, float C, float Vf, int Np) {
    // step response of RCL series circuit
    float wn,csi,tau,step,a,b,tauL;
    float VR, VC, VL;
    int i;
    FILE *fp;
    fp=fopen("RCLstep.txt","w");
    fprintf(fp,"\nStep response of RCL series circuit; final step voltage = %fV\n",Vf);
    fprintf(fp,"R = %g Ohm, C = %gF, L = %gH\n",R,C,L);
    // csi is the damping factor, wn the natural angular frequency
    csi=R*sqrt(C/L)/2, wn=1/sqrt(L*C);
    fprintf(fp,"Damping factor = %f, natural angular frequency = %grad/s\n",csi,wn);
    fprintf(fp,"VR, VC and VL are voltages across Resistor, Capacitor and Inductor, respectively\n");
    fprintf(fp,"t[s]\tVR(t)[V]\tVC(t)[V]\tVL(t)[V]\n");
    if(csi > 1){
        // overdamping condition
        a=wn*(csi-sqrt(csi*csi-1)), b=wn*(csi+sqrt(csi*csi-1));
        tau=1/a, step=5*tau/Np, tauL=L/R;
        for(i=0;i<=Np;i++)
            VR=Vf*(exp(-a*i*step)-exp(-b*i*step))/(tauL*(b-a));
            VC=Vf*(b-a-b*exp(-a*i*step)+a*exp(-b*i*step))/(b-a);
            VL=Vf-VR-VC;
            fprintf(fp,"%g\t%g\t%g\t%g\n",i*step,VR,VC,VL);
    }
    fclose(fp);
} else {
    if (csi==1) {
        // critical damping condition
        tau=1/wn, step=5*tau/Np;
        for(i=0;i<=Np;i++)
            VR=Vf*i*step*exp(-i*step*wn)/(L/R);
            VC=Vf*(1.0-exp(-i*step*wn)-wn*i*step*exp(-i*step*wn));
            VL=Vf-VR-VC;
            fprintf(fp,"%g\t%g\t%g\t%g\n",i*step,VR,VC,VL);
    }
    else {
        // underdamping condition
        a=wn*(csi-sqrt(csi*csi-1)), b=wn*(csi+sqrt(csi*csi-1));
        tau=1/a, step=5*tau/Np, tauL=L/R;
        for(i=0;i<=Np;i++)
            VR=Vf*(exp(-a*i*step)-exp(-b*i*step))/(tauL*(b-a));
            VC=Vf*(b-a-b*exp(-a*i*step)+a*exp(-b*i*step))/(b-a);
            VL=Vf-VR-VC;
            fprintf(fp,"%g\t%g\t%g\t%g\n",i*step,VR,VC,VL);
    }
    fclose(fp);
}
fclose(fp);

else {
    // underdamping condition, csi is less than 1
    tau=1/(wn*csi), step=5*tau/Np, tauL=L/R, a=wn*sqrt(1-csi*csi);
    for(i=0;i<=Np;i++){
        VR=Vf*exp(-wn*csi*i*step)*sin(a*i*step)/(a*tauL);
        VC=Vf*(1.0-exp(-wn*csi*i*step)*sin(a*i*step+acos(csi))*wn/a);
        VL=Vf-VR-VC;
        fprintf(fp,"%g	%g	%g	%g",i*step,VR,VC,VL);
    }
    fclose(fp);
}

}

    float tau,step;
    int i;
    FILE *fp;
    
    fp=fopen("RCramp.txt","w");
    fprintf(fp,"Ramp response of RC circuit; ramp slope = %fV/s",P);
    fprintf(fp,"t[s]	Vo(t)[V]");
    tau=R*C, step=5*tau/Np;
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g	%g",i*step,P*tau*((i*step/tau)-1+exp(-i*step/tau)));
    fclose(fp);
}

void LRramp(float R, float L, float P, int Np){ // ramp response of an LR circuit
    float tau,step;
    int i;
    FILE *fp;
    
    fp=fopen("LRramp.txt","w");
    fprintf(fp,"Ramp response of LR circuit; ramp slope = %fV",P);
    fprintf(fp,"t[s]	Vo(t)[V]");
    tau=L/R, step=5*tau/Np;
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g	%g",i*step,P*tau*((i*step/tau)-1+exp(-i*step/tau)));
    fclose(fp);
}

    float tau,step;
    int i;
    FILE *fp;
    
    fp=fopen("CRramp.txt","w");
    fprintf(fp,"Ramp response of CR circuit; ramp slope = %fV/s",P);
    fprintf(fp,"t[s]	Vo(t)[V]");
    tau=R*C, step=5*tau/Np; // constant time calculation
    for(i=0;i<=Np;i++)
        fprintf(fp,"%g	%g",i*step,P*tau*(1-exp(-i*step/tau)));
    fclose(fp);
}

    float tau,step;
    int i;
    FILE *fp;
    
    fp=fopen("RLramp.txt","w");
    fprintf(fp,"Ramp response of RL circuit; ramp slope = %fV",P);
fprintf(fp,"unt[s]tVo(t)[V]");
tau=L/R, step=5*tau/Np;
for(i=0;i<=Np;i++)
    fprintf(fp,"un%gt%g",i*step,P*tau*(1-exp(-i*step/tau)));
fclose(fp);
}

void CLramp(float L, float C, float P, int Np) {
    // ramp response of CL circuit
    float wn, step;
    int i;
    FILE *fp;
    fp=fopen("CLramp.txt","w");
    fprintf(fp,"Cl ramp response; ramp slope = %fV/s",P);
    fprintf(fp,"unt[s]tVo(t)[V]");
    wn=1/sqrt(L*C), step=4*M_PI/(wn*Np);
    for(i=0;i<=Np;i++)
        fprintf(fp,"un%gt%g",i*step,P*sin(wn*i*step)/wn);
fclose(fp);
}