CHAPTER 3

Design and Development of a Continuous Dilution Technique for the Measurement of Dielectric Constants of Binary Liquid Mixtures: Software Design
Section 3.1
SALIENT FEATURES OF SILABS & KEIL C CROSS COMPILER

In our design, the heart of hardware module is the Cygnal microcontroller which is the
programmable IC. The programming language used for developing the software to the
present microcontroller is an embedded C\textsuperscript{1,2}. The Silicon Laboratories (SiLabs) is an
integrated development environment has become most popular with some special
facilities available with it. SiLabs contains a set of development tools for Cygnal
C8051F020 microcontroller applications. In the present program SiLab IDE with Keil’s\textsuperscript{3}
C cross compiler is used. Micro Flash programmer is used for burning the developed
code in SiLabs in to the microcontroller chip.

SiLab IDE

The SiLabs IDE is a Windows-based software development platform that combines a
robust editor, project manager, and makes facility. The SiLab IDE combines project
management, a rich-featured editor with interactive error correction, option setup, and on-
line help. SiLab is used to create source files and organize them into a project that defines
target application. SiLabs automatically compiles, assembles, and links embedded
application and provides a single focal point for development efforts with the help of Keil
Cross Compiler.

SiLabs helps expedite the development process of embedded applications by providing
the following:

- Full-featured source code editor,
- Device database for configuring the development tool setting,
- Project manager for creating and maintaining projects,
Integrated make facility for assembling, compiling, and linking embedded applications,
Dialogs for all development tool settings,
True integrated source-level Debugger with high-speed CPU and peripheral simulator,
Links to development tools manuals, device datasheets & user's guides.

Software Development Cycle
When we use the SiLabs IDE Software tools, the project development cycle is roughly the same as it is for any other software development project.

1. Create a project, select the target chip from the device database, and configure the tool settings.
2. Create source files in C or assembly.
3. Build your application with the project manager.

Keil Cross Compiler (C51)
Source files are created by the SiLab IDE and are passed to the Keil Cross Compiler C51. The compiler processes source files and create re-locatable object files. The Keil C51 Compiler is a full ANSI implementation of the C programming language that supports all standard features of the C language. In addition, numerous features for direct support of the C8051F020 architecture have been added.

Keil Debugger
The Keil symbolic, source-level debugger is ideally suited for fast, reliable program debugging. The debugger includes a high-speed simulator that let you simulate an entire
C8051F020 system including on-chip peripherals and external hardware. The attributes of the chip we use are automatically configured when we select the device from the Device Database.

**Optimizing Cross Compiler**

The Keil C51 Cross Compiler is an ANSI C Compiler that was written specifically to generate fast, compact code for the 8051 microcontroller family. The C51 Compiler generates object code that matches the efficiency and speed of assembly programming. Using a high-level language like C has many advantages over assembly language programming:

- Knowledge of the processor instruction set is not required. Rudimentary knowledge of the memory structure of the 8051 CPU is desirable (but not necessary).
- Details like register allocation and addressing of the various memory types and data types is managed by the compiler.
- Programs get a formal structure (which is imposed by the C programming language) and can be divided into separate functions. This contributes to source code reusability as well as better overall application structure.
- The ability to combine variable selection with specific operations improves program readability.
- Keywords and operational functions that more nearly resemble the human thought process may be used.
- Programming and program test time is drastically reduced.
Debugging:
The C51 Compiler uses the Intel Object Format (OMF51) for object files and generates complete symbol information. Additionally, the compiler can include all the necessary information such as; variable names, function names, line numbers, and so on to allow detailed and thorough debugging and analysis with the SiLab IDE.

Advantages of Embedded C:
• The assembly code is difficult to read and maintain.
• The amount of code reusable from assembly code is very low.
• C programs are easy to read, understand, maintain, because it possesses greater structure.
• With C the programmer need not know the architecture of the processor.
• Code developed in C will be more portable to other systems rather than in assembly.

Difference between Conventional C and Embedded C:
• Compilers for conventional C are TC, BC
• Compilers for Embedded C are Keil µVision-2 & 3, PIC C etc.
• Conventional C programs need compiler to compile the program & run it.
• The embedded C program needs a cross compiler to compile & generate HEX code.
• The programs in C are basically processor dependent whereas Embedded C programs are micro controller dependent.
• The C program is used for developing an application and not suitable for embedded systems.
• The embedded C is an extension of the conventional C. i.e. Embedded C has all the
features of normal C, but has some extra added features which are not available in C.

- Many functions in C do not support Reentrant concept of functions.
- C is not memory specific. i.e. variables cannot be put in the desired memory location but the location of variable can be found out.
- In embedded C this can be done using specific inbuilt instructions.
- C depends on particular processor or application.
- Embedded C is Controller or target specific.
- Embedded C allows direct communication with memory.

**Why C for Microcontrollers:**

- Compatibility
- Direct access to hardware address
- Direct connection to interrupts
- Optimization consideration
- Development environment
- Reentrancy

**Rules for developing Embedded C Program:**

- Code Optimization.

1. Minimize local variables

If the number of local variables in a function is less, the compiler will be able to fit them into registers. Hence, it will be avoiding frame pointer operations on local variables that are kept on stack. This can result in considerable improvement due to two reasons:

- All local variables are in registers so this improves performance over accessing them
from memory.

- If no local variables need to be saved on the stack, the compiler will not incur the overhead of setting up and restoring the frame pointer.

2. Declare local variables in the inner most scope
- Do not declare all the local variables in the outermost function scope.
- If local variables are declared in the inner most scope.
- If the parameter was declared in the outermost scope, all function calls would have incurred the overhead of object.

- Place case labels in narrow range
- If the case labels are in a narrow range, the compiler does not generate a if-else-if cascade for the switch statement.
- Instead, it generates a jump table of case labels along with manipulating the value of the switch to index the table.
- This code generated is faster than if-else-if cascade code that is generated in cases where the case labels are far apart.
- Also, performance of a jump table based switch statement is independent of the number of case entries in switch statement.

**Reduce the number of parameters**

Function calls with large number of parameters may be expensive due to large number of parameter pushes on stack on each call. For the same reason, avoid passing complete structures as parameters. Use pointers and references in such cases.

**Use references for parameter passing and return value for types bigger than 4 bytes**

Passing parameters by value results in the complete parameter being copied on to the stack.
This is fine for regular types like integer, pointer etc. These types are generally restricted to four bytes.

When passing bigger types, the cost of copying the object on the stack can be prohibitive. When the function exits the destructor will also be invoked.

Thus it is efficient to pass references as parameters. This way it saves on the overhead of a temporary object creation, copying and destruction. This optimization can be performed easily without a major impact to the code by replacing pass by value parameters by const references. (It is important to pass const references so that a bug in the called function does not change the actual value of the parameter). Passing bigger objects as return values also has the same performance issues. A temporary return object is created in this case too.

• Use All the SFR’s in capital letters only.

• Reduce the warnings in the program.

• Make use of MACRO definitions in the program.

• Always define the variables in the code memory by using the keyword code in declaration.

• Eg unsigned int code a[] = { }; 

• Always define as unsigned type of declaration.

• Make use of sbit definition for single bit declaration.

• Eg sbit rs = P3^6;

• Since these are not floating point co-processor, no decimal values can be given as input to them.
• So cannot define the above declaration as sbit rs = P3.6.

• The declaration like this below is invalid.

\[
P3^\wedge 6 = 0;
\]

• P3^6 is bit addressable type & 0 is a 8 bit data which cannot be stored in single bit.

• Permanent termination of the program is got by using while(1);
Section 3.2

SOFTWARE DEVELOPMENT

The software for the development of a continuous dilution technique for the measurement of excess dielectric constant in binary liquid mixtures is developed using KEIL'S C – cross complier. The main role of the software in the present study is to test the activities of the following hardware modules.

1. To measure the frequency of the oscillator.
2. To control the movement of stepper motor and quantity of solute dropped in the dielectric cell
3. To measure the capacitance and dielectric constant.
4. To measure the temperature of the any solution with an accuracy of ± 0.5°C.
5. To display the measured data on LCD Module.

The program codes are stored in the program memory of the Cygnal microcontroller using the JTAG and the program is executed.

The flow chart of the program is presented Fig. 3.1.
Fig. 3.1: Flow chart of the program

START

Initialization of on-chip peripherals
(ADC, Timer, Xbars, Ports etc)

Initialization of LCD

Initialization of program variables, constants (ca, cr, er)

Read the count value through timer 0

Calculate dielectric constant
\[ dc = \frac{(ca-ex)}{(ca-cr)}; \]
\[ ex = 1 + (dc^*(er-1)); \]

Display dielectric constant on LCD module

Read the analog voltage through on-chip ADC (i.e., temperature using LM35)

Display current temperature on LCD module

Fig.3.1:Flow chart of the program
Section 3.3

PROGRAM IN DETAIL

The software developed in the present study is a user-friendly program and easy to operate. The software package is written with a file name **DI-LIQUID.C**. This is written, compiled and executed in the Keil’s C-cross compiler package. The execution of the program gives the dielectric constant of a given liquid. The program details are given below.

**Embedded C Program**

```c
#include<c8051f020.h>
#define SYSCLK 24000000

xdata long int d,d1,d2,d3,d4,d5,d6;

xdata float f;

int flag;

void main()
{
    void lcd_ini(void);
    void lcd_command(void);
    void lcd_data(void);
    void data_conv(void);
    
    #include<e8051f020.h>
    #define SYSCLK 24000000
    xdata long int d,d1,d2,d3,d4,d5,d6;
    xdata float f;
    int flag;
    void main()
    {
        void lcd_ini(void);
        void lcd_command(void);
        void lcd_data(void);
        void data_conv(void);
        
#include<e8051f020.h>
#define SYSCLK 24000000
xdata long int d,d1,d2,d3,d4,d5,d6;
xdata float f;
int flag;
void main()
{
void lcd_ini(void);
void lcd_command(void);
void lcd_data(void);
void data_conv(void);

```
void diel_display(void);
void temp_display(float);
void osc_init(void);
void freq_isr(void);
void delay(void);
void delay1(void);
void adc_init(void);
float adc_conv(void);
void temp_disp(float);
int i,j;
long int hc,lc,T;
float C,v,v1;
int ml;
unsigned int diel[] = {'D','i','e','l','o','n'};
int temp[] = {'T','e','m','p','s'};
int F_code[] = {0x09,0x0a,0x06,0x05};
int B_code[] = {0x05,0x06,0x0a,0x09};
float de, cx, ea = 227.0, cr = 277.5, er = 5.91, ex;
WDTCN = 0xde;
WDTCN = 0xad;
XBR0 = 0xef;
XBR1 = 0xff;
XBR2 = 0x5d;
P74OUT = 0x04;
P1MDOUT = 0x10;
osc_init();
lcd_init();
adc_init();
delay();
P4 = 0x01;

cmd_comand();
delay();
PCA0MD = 0x00;
PCA0CPM2 = 0x46;
PCA0CN = 0x00;
PCA0CPL2 = 0x00;
PCA0CPH2 = 0x0a;
CR = 1;
flag = 0;
CKCON = 0x00;
TMOD = 0x55;
TH1 = 0xd8;
TL1 = 0xf0;
TH0 = 0x00;
TL0 = 0x00;
TCON = 0x50;
IE = 0x88;
ml = 0;
do {
while(flag == 0);
flag = 0;
lc = TL0;
hc = TH0;
T = hc*256+lc;
v = T*10;
C = 33000/v;
f = C * 1000;
ex = f;
dc = (ca-cx)/(ca-cr);
f = dc;
ex = 1+(dc*(er-1));
f = ex*1000;
data_conv();
P4 = 0x02;
lcd_comand();
for(i=0;i<8;i++)
{
P4 = diel[i];
lcd_data();
}
diel_display();
P4 = 0xc0;
lcd_command();
for(i=0;i<=4;i++)
{
P4 = temp[i];
lcd_data();
}
v1 = adc_conv();
temp_display(v1);
delay1();
for(i=0;i<360;i++)
{
for(j=0;j<4;j++)
{
P5 = F_code[j];
delay();
}
}
TMOD = 0x55;
TH1 = 0xd8;
TL1 = 0xf0;
TH0 = 0x00;
TL0 = 0x00;
TCON = 0x50;
IE = 0x88;
ml++;
}
while (ml<5);

for( i=0;i<1800;i++)
{
for(j=0;j<4;j++)
{
P5 = B_code[j];
delayO;
}
}
delay1();
while(1);

void adc_init(void)
{
REFOCN = 0x03;
AMX0CF = 0x00;
AMX0SL = 0x03;
ADC0CF = 0x40;
ADC0CN = 0x00;
ADOEN = 1;
}
float adc_conv()
{
    float v,vx,lb,hb,sum=0.0;
    int i,x;
    for(i=1; i<=5000; i++)
    {
        AD0INT = 0;
        AD0BUSY = 1;
        while(AD0INT !=1);
        lb = ADC0L;
        hb = ADC0H;
        x = hb*256 + lb;
        v = (x*2.4)/4096;
        sum = sum+v;
    }
    vx = sum/5000;
    return vx;
}
void temp_display(float disp)
{
    disp = disp*1000;
    d = disp;
    d1 = d/100;
    d1 = d1+0x30;
    P4 = d1;
    lcd_data();
    d2 = d % 100;
    d2 = d2/10;
    d2 = d2+0x30;
    P4 = d2;
    lcd_data();
    P4 = 0x2e;
    lcd_data();
    d3 = d % 10;
    d3 = d3+0x30;
    P4 = d3;
    lcd_data();
}

void lcd_ini(void)
{  
P4 = 0x38;
  lcd_comand();
  P4 = 0x0e;
  lcd_comand();
  P4 = 0x06;
  lcd_comand();
  P4 = 0x01;
  lcd_comand();
  P4 = 0x02;
  lcd_comand();
}

void lcd_comand(void)
{
  int j;
  P7 = 0x80;
  for(j=0;j<10000;j++);
  P7 = 0x00;
  for(j=0;j<10000;j++);
}

void lcd_data(void)
{
  int k;

P7 = 0xc0;
for(k=0;k<10000;k++);
P7 = 0x40;
for(k=0;k<10000;k++);
}

void data_conv(void)
{

d = f;
d1 = d/100000;
d2 = d % 100000;
d2 = d2/10000;
d3 = d % 10000;
d3 = d3/1000;
d4 = d % 1000;
d4 = d4/100;
d5 = d % 100;
d5 = d5/10;
d6 = d % 10;
}

void display(void)
{

d1 = d1 +0x30;
P4 = d1;
lcd_data();
d2 = d2 +0x30;
P4 = d2;
lcd_data();
d3 = d3 +0x30;
P4 = d3;
lcd_data();
P4 = 0x2e;
lcd_data();
d4 = d4 +0x30;
P4 = d4;
lcd_data();
d5 = d5 +0x30;
P4 = d5;
lcd_data();
}

void delay(void)
{
    int l,k;
    for(l = 0; l<50; l++)
        for (k = 0; k<1000; k++);
}

void delay1(void)
{
    int r1,r2;
    for(r1 = 0; r1<10000; r1++)
    for (r2 = 0; r2<2000; r2++);
}

void freq_isr(void) interrupt 3
{
    TCON = 0x00;
    IE = 0x00;
    flag = 1;
}

void osc_init(void)
{
    int i;
    OSCXCN = 0x67;
    for (i=0; i <256; i++);
    while (!(OSCXCN & 0x80));
    OSCICN = 0x88;
}
Section 3.4

CALIBRATION AND MEASUREMENT PROCEDURE

The procedure for calibration of the system and measurement of dielectric constant of pure liquids is mentioned below:

1. Clean the dielectric cell, dry it and keep it in a beaker containing air.
2. Connect the cell to the circuit as shown in Figs 2.1 and 2.21.
3. Switch on the system and activate the software.
4. The system measures and displays the frequency along with temperature and in turn the capacitance of the cell using the equation (2). Store the values.
5. Keep the reference liquid (cyclohexane in the present study) in the cell.
6. Repeat the steps from (2) to (4).
7. Keep the unknown liquid in the cell.
8. Repeat the steps from (2) to (4).
9. Calculate the dielectric constant of unknown liquid using the equation (2).
10. Record the readings of the dielectric constant of unknown liquids along with the temperature.

The procedure for calibration of the system and measurement of dielectric constants of binary liquid mixtures is given below:

1. Clean the dielectric cell, dry it and keep it in a beaker containing air.
2. Keep the cell in a temperature bath which is maintained constant at the required temperature.
3. Connect the cell to the circuit as shown in Figs 2.1 and 2.21.

4. Switch on the system and activate the software.

5. The system measures and displays the frequency along with temperature and in turn the capacitance of the cell using the equation (2.2). Store the values.

6. Keep the reference liquid (cyclohexane in the present study) in the cell.

7. Repeat the steps from (2) to (5).

8. Keep the solvent of known mass \( m_2 \) in the dielectric cell.

9. Keep the solute in the syringe.

10. Drop the solute in the solvent with the help of stepper motor on rotating through the fixed number of steps which decides the mass of the solute.

11. Mix the contents of the cell with the magnetic stirrer.

12. Repeat the step (5).

13. Determine the mole fraction of the solute added to the solvent using equation (2.4).

14. Calculate the dielectric constant of binary liquid mixture using the equation (2.2).

15. Store/record the readings of the dielectric constant of binary liquid mixture along with temperature.

16. Repeat the steps from (10) to (15) till the solute is emptied in the syringe.

The system is calibrated with the standard pure liquids on measuring their dielectric constants at 30°C as per the procedure mentioned above. The samples are selected such that a wide range is covered. The results of the measurements are presented in Table 3.1. The results of the present study are in good agreement with the literature values with an accuracy of ± 0.2% as shown in the Table.
Table 3.1

Dielectric constants of pure liquids at 30°C

<table>
<thead>
<tr>
<th>Sample</th>
<th>Present work</th>
<th>Literature</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Toulene</td>
<td>2.45</td>
<td>2.30</td>
<td>4</td>
</tr>
<tr>
<td>Acetone</td>
<td>20.29</td>
<td>20.35</td>
<td>5</td>
</tr>
<tr>
<td>Methanol</td>
<td>32.56</td>
<td>32.60</td>
<td>4</td>
</tr>
<tr>
<td>Acetonitrile</td>
<td>35.09</td>
<td>35.20</td>
<td>4</td>
</tr>
<tr>
<td>Cyclohexane</td>
<td>2.09</td>
<td>2.16</td>
<td>4</td>
</tr>
<tr>
<td>Carbontetrachloride</td>
<td>2.21</td>
<td>2.15</td>
<td>6</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.25</td>
<td>7</td>
</tr>
</tbody>
</table>
REFERENCES


5. The website www.asiinstr.com. This site for dielectric constant tables.

