CHAPTER 3
3.1. INTRODUCTION

In the present work to program the LPC2129 ARM7 processor the Keil μVision version 4 software is used. The Keil μVision is an Integrated Development Environment (μVision IDE) that supports three major microcontroller architectures and sustains the development of a wide range of applications [45].

(i).8-bit (classic and extended 8051) devices include an efficient interrupt system designed for real-time performance and are found in more than 65% of all 8-bit applications. Over 1000 variants are available, with peripherals that include analog I/O, timer/counters, PWM, serial interfaces like UART, I2C, SPI, USB, CAN, and on-chip RF transmitter supporting low-power wireless applications. Some architecture extensions provide up to 16MB memory with an enriched 16/32-bit instruction set. The μVision IDE supports the latest trends, like custom chip designs based on IP cores, which integrate application-specific peripherals on a single chip.

(ii).16-bit (Infineon C166, XE166, XC2000) devices are tuned for optimum real-time and interrupt performance and provide a rich set of on-chip peripherals closely coupled with the microcontroller core. They include a Peripheral Event Controller (similar to memory-to-memory DMA) for high speed data collection with little or no microcontroller overhead. These devices are the best choice for applications requiring extremely fast responses to external events.

(iii).32-bit (ARM7 and ARM9 based) devices support complex applications, which require greater processing power. These cores provide high-speed 32-bit arithmetic within a 4GB address space. The RISC instruction set has been extended with a Thumb mode for high code density. ARM7 and ARM9 devices provide separate stack spaces for high-speed context switching enabling efficient multi-tasking operating systems. Bit addressing and dedicated peripheral address spaces are not supported. Only two interrupt priority levels, - Interrupt Request (IRQ) and Fast Interrupt Request (FIQ), are available [46].
μVision for Windows™ is an Integrated Development Environment that combines project management, source code editing, and program debugging in one single, powerful environment.

3.2 Development Tools

In fact, the Keil Development Tools are designed for the professional software developer, however programmers of all levels can use them to get the most out of the embedded microcontroller architectures that are supported. In addition to the software packages, Keil offers a variety of evaluation boards, USB-JTAG adapters, emulators, and third-party tools, which completes the range of products.

3.2.1. Software Development Tools

Like all software based on Keil's μVision IDE, the toolsets provide a powerful, easy to use and easy to learn environment for developing embedded applications. They include the components required to create, debug, and assemble the C/C++ source files, and incorporate simulation for microcontrollers and related peripherals. The RTX RTOS Kernel helps to implement complex and time-critical software.

![Software Development Tools](image)

Fig.3.1. Software development tools.
3.2.2. RTOS and Middleware Components

These components are designed to solve communication and real-time challenges of embedded systems. While it is possible to implement embedded applications without using a real-time kernel, a proven kernel saves time and shortens the development cycle. This component also includes the source code files for the operating system.

![RTOS and Middleware Components Diagram]

3.2.3. Hardware Debug Adapters:

The µVision Debugger fully supports several emulators provided by Keil, and other vendors. The Keil ULINK USB-JTAG family of adapters connect the USB port of a PC to the target hardware. They enable you to download, test, and debug your embedded application on real hardware [47].
μVision IDE

The μVision IDE is a window-based software development platform combining a robust editor, Project Manager, and Make Utility tool. μVision supports all the Keil tools including C/C++ Compiler, Macro Assembler, Linker, Library Manager, and Object-HEX Converter. μVision helps expedite the development process by providing.

μVision Device Database

The μVision Device Database offers a convenient way to select and configure your device and project parameters. It includes preconfigured settings, so that you can fully concentrate on your application requirements. In addition, you can add your own devices, or change existing settings. Use the features of the Device Database to:

- Initialize the start up code and device settings
- Load the configuration options for the assembler, compiler, and linker
- You can add and change microcontroller configuration settings

3.3. μVision Debugger

The μVision Debugger is completely integrated into the μVision IDE. It provides the following features:

- **Disassembly** of the code on C/C++ source- or assembly-level with program execution in various stepping modes and various view modes, like assembler, text, or mixed mode
- Multiple breakpoint options including access and complex breakpoints **Bookmarks** to quickly find and define your critical spots
- **Review and modify** memory, variable, and register values
• List the program call tree including stack variables
• Review the status of on-chip microcontroller peripherals
• Debugging commands or C-like scripting functions
• Execution Profiling to record and display the time consumed, as well as the cycles needed for each instruction
• Code Coverage statistics for safety-critical application testing
• Various analyzing tools to view statistics, record values of variables and peripheral I/O signals, and to display them on a time axis
• Instruction Trace capabilities to view the history of executed instructions
• Define personalized screen and window layouts.

The µVision Debugger offers two operating modes—Simulator Mode and Target Mode.

Simulator Mode configures the µVision Debugger as a software-only product that accurately simulates target systems including instructions and most on-chip peripherals. In this mode, you can test your application code before any hardware is available. It gives you serious benefits for rapid development of reliable embedded software.

The Simulator Mode offers:

- Software testing on your desktop with no hardware environment
- Early software debugging on a functional basis improves software reliability
- Breakpoints that are impossible with hardware debuggers
- Optimal input signals. Hardware debuggers add extra noise
- Single-stepping through signal processing algorithms is possible. External signals are stopped when the microcontroller halts.
- Detection of failure scenarios that would destroy real hardware peripherals

Target Model connects the µVision Debugger to real hardware. Several target drivers are available that interface to a:

ULINK JTAG/OCDS Adapter that connects to on-chip debugging systems

Monitor that may be integrated with user hardware or that is available on many evaluation boards

Emulator that connects to the microcontroller pins of the target hardware

In-System Debugger that is part of the user application program and provides basic test functions
**ULINKPro Adapter** a high-speed debug and trace unit connecting to on-chip debugging systems via JTAG/SWD/SWV, and offering Cortex-M3 ETM Instruction Trace capabilities.

### 3.4. Assembler

An assembler allows you to write programs using microcontroller instructions. It is used where utmost speed, small code size, and exact hardware control is essential. The Keil Assemblers translate symbolic assembler language mnemonics into executable machine code while supporting source-level symbolic debugging. In addition, they offer powerful capabilities like macro processing. The assembler translates assembly source files into re-locatable object modules and can optionally create listing files with symbol table and cross-reference details. Complete line number, symbol, and type information is written to the generated object files. This information enables the debugger to display the program variables exactly. Line numbers are used for source-level debugging with the μVision Debugger or other third-party debugging tools.

Keil assemblers support several different types of macro processors (depending on architecture):

The **Standard Macro Processor** is the easier macro processor to use. It allows you to define and use macros in your assembly programs using syntax that is compatible with that used in many other assemblers.

The **Macro Processing Language or MPL** is a string replacement facility that is compatible with the Intel ASM-51 macro processor. MPL has several predefined macro processor functions that perform useful operations like string manipulation and number processing. Macros save development and maintenance time, since commonly used sequences need to be developed once only.

Another powerful feature of the assembler’s macro processor is the conditional assembly capability. You can invoke conditional assembly through command line directives or symbols in your assembly program. Conditional assembly of code sections can help achieve the most compact code possible. It also allows you to generate different applications from a single assembly source file.

### 3.5. C/C++ Compiler

The ARM C/C++ compiler is designed to generate fast and compact code for the ARM7, ARM9 and Cortex-Mx processor architectures; while the Keil ANSI C-compilers target the 8051, C166, XE166, and XC2000 architectures. They can generate object code that matches the efficiency
and speed of assembly programming. Using a high-level language like C/C++ offers many advantages over assembly language programming.

❖ Knowledge of the processor instruction set is not required. Rudimentary knowledge of the microcontroller architecture is desirable, but not necessary.
❖ Details, like register allocation, addressing of the various memory types, and addressing data types, are managed by the compiler.
❖ Programs receive a formal structure (imposed by the C/C++ programming language) and can be split into distinct functions. This contributes to source code reusability as well as a better application structure.
❖ Keywords and operational functions that resemble the human thought process may be used.
❖ Software development time and debugging time are significantly reduced.
❖ You can use the standard routines from the run-time library such as formatted output, numeric conversions, and floating-point arithmetic.
❖ Through modular programming techniques, existing program components can be integrated easily into new programs.
❖ The C/C++ language is portable (based on the ANSI standard), enjoys wide and popular support, and is easily obtained for most systems. Existing program code can be adapted quickly and as needed to other processors.

3.6. Object-HEX Converter

The object-hex converter creates Intel HEX files from absolute object modules that have been created by the linker. Intel HEX files are ASCII files containing a hexadecimal representation of your application program. They are loaded easily into a device program for writing to ROM, EPROM, FLASH, or other programmable memory. Intel HEX files can be manipulated easily to include checksum or CRC data.

3.6.1. Linker/Locator

The linker/locator combines object modules into a single, executable program. It resolves external and public references and assigns absolute addresses to re-locatable program segments. The linker includes the appropriate run-time library modules automatically and processes the object modules created by the Compiler and Assembler. You can invoke the linker from the command line or from within the μVision IDE. To accommodate most applications, the default...
linker directives have been chosen carefully and need no additional options. However, it is easy to specify additional custom settings for any application.

3.6.2. Library Manager
The library manager creates and maintains libraries of object modules (created by the C/C++ Compiler and Assembler). Library files provide a convenient way to combine and reference a large number of modules that may be used by the linker. The linker includes libraries to resolve external variables and functions used in applications. Modules from libraries are extracted and added to programs only if required. Modules, containing routines that are not invoked by your program specifically, are not included in the final output. Object modules extracted by the linker from a library are processed exactly like other object modules. There are a number of advantages to using libraries: security, speed, and minimized disk space are only a few. Libraries provide a vehicle for distributing large numbers of functions and routines without distributing the original source code. For example, the ANSI C library is supplied as a set of library files. You can build library files (instead of executable programs) using the μVision Project Manager. To do so, check the Create Library check box in the Options for Target — Output dialog. Alternatively, you may invoke the library manager from the Command Window.

3.7. μVision Modes
μVision operates in two modes: Build Mode and Debug Mode. Screen settings, Toolbar settings, and project options are stored in the context of the mode. The File Toolbar is enabled in all modes, while the Debug Toolbar and Build Toolbar display in their respective mode only. Buttons, icons, and menus are enabled if relevant for a specific mode. The standard working mode is Build Mode. In this mode you write your application, configure the project, set preferences, select the target hardware and the device; you will compile, link, and assemble the programs, correct the errors, and set general settings valid for the entire application.

In Debug Mode, you can also change some general options and edit source code files, but these changes will only be effective after you have switched back to Build Mode, and rebuild your application. Changes to debug settings are effective immediately.

3.7.1. Creating Embedded Programs
μVision is a Windows application that encapsulates the Keil microcontroller development tools as well as several third-party utilities. μVision provides everything you need to start creating
embedded programs quickly. µVision includes an advanced editor, project manager, and make utility, which work together to ease your development efforts, decreases the learning curve, and helps you to get started with creating embedded applications quickly [49].

There are several tasks involved in creating a new embedded project:

i. Creating a Project File
ii. Using the Project Windows
iii. Creating Source Files
iv. Adding Source Files to the Project
v. Using Targets, Groups, and Files
vi. Setting Target Options, Groups Options, and File Options
vii. Configuring the Startup Code
viii. Building the Project
ix. Creating a HEX File
x. Working with Multi-Projects.

3.7.2. Creating a Project File

Creating a new µVision project requires just three steps:

1. Select the Project Folder and Project Filename
2. Select the Target Microcontroller
3. Copy the Startup Code to the Project Folder.

3.7.3. Selecting the Folder and Project Name

To create a new project file, select the Project – New Project… Menu. This opens a standard dialog that prompts you for the new project file name. It is good practice to use a separate folder for each project. You may use the Create New Folder button in this dialog to create a new empty folder. Select the preferred folder and enter the file name for the new project. µVision creates a new, empty project file with the specified name. The project contains a default target and file group name, which you can view on the Project Window.

3.7.4. Selecting the Target Microcontroller

After you have selected the folder and decided upon a file name for the project, µVision asks you to choose a target microcontroller. This step is very important, since µVision customizes the tool settings, peripherals, and dialogs for that particular device. The Select Device dialog box
lists all the devices from the μVision **Device Database**. You may invoke this screen through the **Project – Select Device for Target**...Menu in order to change target later.

![Select Device for Target](image)

**Fig. 3.4. Window for select device for target**

3.7.5. Creating Source Files

Use the button on the **File Toolbar** or the select the **File – New...** Menu to create a new source file. This action opens an empty **Editor Window** to enter your source code. MVision enables color syntax highlighting based on the file extension (after you have saved the file). To use this feature immediately, save the empty file with the desired extension prior to starting coding. Save the new source file using the button on the **File Toolbar** or use the **File – Save** Menu.

3.7.6. Adding Source Files to the Project

After creating and saving the source file, it should be added to the project. Files existing in the project folder, but not included in the current project structure, will not be compiled. Right-click a file group in the **Project Window** and select **Add Files to Group** from the **ContextMenu**.
Then, select the source file or source files to be added. A self-explanatory window will guide you through the steps of adding a file.

**Fig. 3.5. Add files to Current project window**
3.7.7. Setting Target Options

Open the Options for Target dialog from the Build Toolbar or from the Project Menu.

Through this dialog box, you can:

- change the target device
- set target options
- and configure the development tools and utilities

Normally, you do not have to make changes to the default settings in the Target and Output dialog. The options available in the Options for Target dialogs depend on the microcontroller device selected. Of course, the available tabs and pages will change in accordance with the device selected and with the target. When switching between devices, the menu options are available as soon as the OK button in the Device Selection dialog has been clicked.

3.7.8. Creating a HEX File

Check the Create HEX File box under Options for Target — Output, and μVision will automatically create a HEX file during the build process. Select the desired HEX format through...
the drop-down control to generate formatted HEX files, which are required on some Flash programming utilities.

![Fig. 3.7. Target simulator options window.](image)

3.8. Debugging

The μVision Debugger can be configured as a Simulator or as a Target Debugger. Go to the **Debug** tab of the **Options for Target** dialog to switch between the two debug modes and to configure each mode. The **Simulator** is a software-only product that simulates most features of a microcontroller without the need for target hardware. By using the Simulator, you can test and debug your embedded application before any target hardware or evaluation board is available. μVision also simulates a wide variety of peripherals including the serial port, external I/O, timers, and interrupts. Peripheral simulation capabilities vary depending on the device you have selected.
The Target Debugger is a hybrid product that combines μVision with a hardware debugger interfacing to your target system.

The following debug devices are supported:

- **JTAG/OCDS Adapters** that connect to on-chip debugging systems like the ARM Embedded ICE
- **Target Monitors** that are integrated with user hardware and that are available on many evaluation boards
- **Emulators** that connect to the MCU pins of the target hardware
- **In-System Debuggers** that are part of the user application program and provide basic test functions.

To debug programs using the Simulator, check **Use Simulator** on the left side of the Debug dialog. To debug programs running on target hardware, check **Use <Hardware Debugger>** on the right side of the Debug dialog.

---

**Fig. 3.8. Options for measure window**
In addition to selecting whether you debug with the Simulator or Target Debugger, the Debug dialog provides a great variety of debugger configuration options.

3.9. Simulation

μVision simulates up to 4 GB of memory from which specific areas can be mapped for reading, writing, executing, or a combination of these. In most cases, μVision can deduce the correct memory map from the program object module. Any illegal memory access is automatically trapped and reported. A number of device-specific simulation capabilities are possible with μVision. When you select a microcontroller from the Device Database, μVision configures the Simulator accordingly and selects the appropriate instruction set, timing, and peripherals.

The μVision Simulator:

- Runs programs using the ARM7, ARM9, Thumb, Thumb2, 8051, C166/XE166/XC2000 instruction sets.
- Is cycle-accurate and correctly simulates instructions and on-chip peripheral timing, where possible
- Simulates on-chip peripherals of many 8051, C166/XE166/XC2000, ARM7, ARM9, and Cortex-Mx devices
- Can provide external stimulus using the debugger C script language.

3.9.1 Starting a Debug Session

When you start a debug session, μVision loads the application, executes the startup code, and, if configured, stops at the main C function. When program execution stops, μVision opens a Text Editor window, with the current source code line highlighted, and a Disassembly Window, showing the disassembled code. Use the Start/Stop Debug Session command of the Debug Toolbar to start or stop a debugging session. Screen layouts are restored when entering and saved automatically when closing the Debugger. The current instruction or high-level statement (the one executed on the next instruction cycle) is marked with a yellow arrow. Each time you step, the arrow moves to reflect the new current line or instruction.

3.10. Executing Code

μVision provides several ways to run your programs. You can instruct the program to run directly to the main C function. Set this option in the Debug tab of the Options for Target dialog.

- Select debugger commands from the Debug Menu or the Debug Toolbar
Enter debugger commands in the **Command Window**

Execute debugger commands from an initialization file.

This hex file will be downloaded into the LPC2378 controller by Philips LPC 2000 Flash Magic Software [50].

**Steps to use the Flash magic Software**

1. First run the Flash magic Application, and the following window will appear.

![Flash Magic](image)

**Fig.3.9. NXP Flash Magic**

2. In Step1 section, Select COM Port as COM1, Baud Rate as 19200, Device as LPC2129, Interface as NXP ICP Bridge and Oscillator frequency (MHz) as 12.0000.

3. Enable Erase Blocks used by hex file for erasing the memory locations of microcontroller before programming, in Step 2 section.

4. Click the Browse Button, for Selecting the Hex file to be downloaded in Step 3 section.

5. Enable Verify, After Programming Option in Step 4 section, for verifying data's in memory locations after programming success.
6. Select Start button in Step 5 section, for hex file downloaded to the LPC2129 controller.

7. If given Hex file is properly downloaded, the window is as shown in below

![Flash Magic Window for options](image)

### 3.10. Flash Magic Window for options

8. The downloaded Program will start after changing the Mode of Execution in the board (Change the Toggle switch to Execution Mode and Press the Reset key).

### 3.11. Software Program (Embedded C Program)

The embedded C program developed in the present design to display the data on the LCD module and also to interface the mobile using the GSM modem is given below. This program is developed using the Keil μVision4 C compiler.

```c
#include <LPC21xx.H>
```

54
#include "adc.h"
#include "lcd.h"
#include "serial1.h"
#include "common.h"

void newline(void);

unsigned char
h_val1=10,h_val2=20,h_val3=30,h_val4=40,h_val5=50,h_val6=60,h_val7=70,h_val8=80,h_val9=90;

int main(void)
{
    unsigned int adcVal1,adcVal2,adcVal3;
    unsigned char *ptr1,*ptr2,*ptr3;

    IODIR0=0X00000080;
    IOCLR0=0X00000080;
    setClock();
    InitSerial1(9600);
    adcInit();
    putStrS1("AT");
    newline();
    delay(10);
    putStrS1("AT+CMGF=1");
    newline();
delay(10);
putStrS1("AT+CREG=1");
newline();
delay(10);
lcdInit();
putStrL("Lcd Test" ,0x01);
putStrL("",0x01);
while(1) {
putStrS1("AT+CMGS=");
putCharS1(""");
putStrS1("9440160099");
putCharS1(""");
newline();
putStrS1("FOR YOUR ATTENTION PLEASE ");
newline();

/ *---------**CO2 GAS SENSOR **---------*/
putStrL("CO2:" ,0x80);
adcVal1 = captureADC(CH0);
ptr1 = getBCD(adcVal1+200);
putStrL(ptr1,0x84);
putStrS1("CO2 = ");
if(adcVal1>400)
{
    IOSET0=0X00000080;
}

putStrS1(ptrl);
putStrS1(" ppm ");
newlineQ;
delay(50);

/*---------**HUMILITY**------------*/
putStrL("RH:",0x89);
adcVal2 = captureADC(CH1);

IOCLR0=0X00000080;

if(adcVal2<=113)
{
    ptr2 = getBCD(h_val1);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
else if((adcVal2>113) && (adcVal2<=226))
{
    ptr2 = getBCD(h_val2);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
else if((adcVal2>226) && (adcVal2<=339))
{
    ptr2 = getBCD(h_val3);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
else if((adcVal2>339) && (adcVal2<=452))
{
    ptr2 = getBCD(h_val4);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrSl("Humidity RH =");
    putStrSl(ptr2);
    putStrSl("%");
    newline();
}
else if((adcVal2>452) && (adcVal2<=565))
{
    ptr2 = getBCD(h_val5);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
else if((adcVal2>565) && (adcVal2<=678))
ptr2 = getBCD(h_val6);
putStrL(ptr2,0x8b);
putStrL("%",0x8f);
putStrS1("Humidity RH =");
putStrS1(ptr2);
putStrS1("%");
newline();

IOSET0=0X00000080;

else if((adcVa!2>678) && (adcVal2<=791))
{
    ptr2 = getBCD(h_val7);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
else if((adcVa!2>791) && (adcVal2<=904))
{
    ptr2 = getBCD(h_val6);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
else if((adcVa!2>791) && (adcVal2<=904))
{ 
    ptr2 = getBCD(h_val8);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}

else if((adcVal2>904) && (adcVal2<=1023))
{
    ptr2 = getBCD(h_val9);
    putStrL(ptr2,0x8b);
    putStrL("%",0x8f);
    putStrS1("Humidity RH =");
    putStrS1(ptr2);
    putStrS1("%");
    newline();
}
delay(50);
/*--------**TEMPERATURE**--------* /

putStrL("Temp:",Oxc0);

adcVal3 = captureADC(CH2);
delay(1000);

adcVal3 = captureADC(CH2);
adcVal3=(adcVal3 * 3.22)/10;

ptr3 = getBCD1(adcVal3);
putStrL(ptr3,0xc9);
putStrL("C",Oxcc);
putStrSl("Temperature=");
if(adcVal3 > 40)
{
    IOSET0=0X00000080;
}
putStrS1(ptr3);
putStrS1("C");
nnewline();
putCharS1(CTRL_Z);
delay(4000);
nnewline();
nnewline();
void newline(void) {
    putCharS1(0x0D);
    putCharS1(0x0A);
}

------------------xx-------------------