4.1 TRMC Software Design

The software implementation of the TRMC (Transmitter Remote Monitoring and Controller) system is developed using embedded C, ASP.NET, C# and SQL database and IDEs are Visual Studio 2010 and Keil-4.

Embedded C software implementation is in embedded target board with ARM Cortex-M3 LM3S9B96 processor. The front-end i.e., webpage has been developed using ASP.NET, a Microsoft server-side web technology. Webpage TCP/IP client and background controls are implemented using C#, an object-oriented programming language of the .Net family from Microsoft.

The sensors’ data archive along with a time stamp is performed by Microsoft SQL Server 2008, a relational database management system. Visual Studio 2010 (VS) IDE (integrated development environment) a set of tools in a single application is used for writing ASP.NET, C# programs. ARM Keil-4 IDE is used to implement embedded ‘C’ program logic on ARM Cortex-M3 processor.

Three software modules execute concurrently for monitoring and controlling MST Radar Transmitters. The MCU software receives analog signals from transmitter, frames digitized data into ethernet packets for transmission to the web server. The web server receives the digital data from the MCU using the TCP/IP protocol and displays on the web page with .NET framework and web application. The SQL database server stores the sensors digitized data for future analysis which is running on the web server. A common web browser acts as a client. Figure 4.1 presents the block diagram of software implementation. The functional description of each module is given below.
4.2 Software Modules

![Functional block diagram of TX monitoring and control system](image)

**Figure 4.1:** Functional block diagram of TX monitoring and control system

The system software is divided into three modules. The first module is firmware code for analog to digital conversion, GPIOs handling and ethernet controller routines that are executed in the ARM Cortex-M3 processor. The second module is for web page designing and TCP/IP server implementation on the remote computer. The third module is routines for database handling and storing data in the database server.
4.2.1 Microcontroller Module

The module contains the software burnt into the Cortex-m3 microcontroller of the motherboard. This program reads sensor data that are PDR, DR, HPA, and Airflow obtained from transmitter and converted into digital form and stored in a buffer dynamically. This program carries out the proper calibration before framing packet. This program reads the values (Tx parameters) from transmitters and converted into digital form and frame a packet sends to the web server through TCP/IP. Software implemented in C language is also responsible for initializing and setting the clock frequency to run directly from the crystal. The internal peripherals enable for using the user application. It also calls conversion routines for calibration conversion of binary values generated by ADC to ASCII values and send to ethernet routines.

It calls delay routines to avoid overlapping such as calibration of sensor values and reading sensor values to achieve good performance. The program is also responsible for operation of transmitter through relay control of all modules viz., solid-state amplifier followed by three stages of triode based amplifiers namely pre-driver, driver, and high power amplifiers. It reads user commands from a web browser where as the program controls the appropriate relays by setting flags. The ISR is invoked whenever data arrives at ethernet port and sets an appropriate flag after storing the data in a buffer. Then the microcontroller sends control data through its appropriate pins for switching relays ON/OFF.

LM3S9B96 microcontroller has a high speed inbuilt two 10-bit 16 channels ADC. It is 12 channels are used for three stages triode based amplifier sensors in the transmitter. This microcontroller after conversion of analog data into digital form
frames the sensor data. It establishes ethernet functionality and sends the packet after receiving the request from a web server running on the personal computer. It also processes the commands or requests received from the web server. After receiving the commands, the microcontroller takes the action on the relevant Tx controls, like switch ON/OFF the Tx anode supplies in the transmitter.

4.2.2 Web Server Module

Web server module contains software for providing web services. It consists of IIS web server with .NET framework and web application, database server and browser as a client. The web application is implemented in C# (C-Sharp). It sends requests to the database server to send updated data for sending the same to a client as a request from the client through its browser. The client may send HTTP requests to the web server for getting updated values of parameters and the control input comes from the user or operator. Then server side web server sends a request to the embedded web server for retrieving updated values from the database. The embedded web server takes the request and sends the updated values to the web server running on the personal computer. The web server in the form of an HTTP response sends same to the client. The application program running at web server initiates necessary control by looking at the changed values in the database server. The advantage provided by such a web service is that the user can view and change the necessary parameter data from any location by authorized users.

The performance of the system from the remote client depends on the network speed, which is directly dependent on allotted bandwidth and congestion level of route. In case of very busy network, if required bandwidth is dedicated seamless and online performance will be observed.
ASP.NET is used to create web pages and web services and is an integral part of Microsoft .NET vision. It takes an object-oriented programming approach to web page execution. Every element in an ASP.NET page is treated as an object and run on the server. On this page code gets compiled into an intermediate language by a .NET common language runtime-compliant compiler. A JIT compiler turns the intermediate code to native machine code, and that machine code is eventually run on the processor. Because of the code runs straight from the processor, pages load much faster than classic ASP pages, where embedded VBscript or Javascript had to be continuously interpreted and cached.

The web server running on the personal computer takes sensor data in the form of packets. The packet splits into parameter wise and displays on web page continuously. The background task whose control and monitor operations is done by using C# [3] (C-Sharp) language.

4.2.3 Database Module

This software runs synchronously with monitoring and controlling. It runs on an embedded web server where the database server is implemented in MS-SQL Server 2008. This application program is implemented in visual studio 2010 and interacts with the database server. It monitors continuously sensed parameters in calibrated form. This software retrieves the sensed data of the parameters and stores the same in appropriate date and time. It also stores the user entered set of values in a separate table based on which parameter control action is initiated by the user. This software enables its user to generate reports on record parameters for particular time and days as well as generate graphs of the same, based on choice of user in the interactive general web browser.
The Microsoft SQL server 2008 is a relational database management system. As a database, it has a software function to store and retrieve data as requested by ASP.NET front end application where both are running on the same computer. It supports managing XML data, as well as relational data. SQL Server 2008 [2] has enhanced with new indexing algorithms, syntax and better error recovery systems. The data pages are check summed for better error resiliency and optimistic concurrency support has added for better performance. The permissions and access control have been made for more granularities and the query processor handles concurrent execution of queries in a more efficient way. The partitions on the tables and indexes are supported natively so that scaling out a database into a cluster is easier. The SQL CLR [7] is introduced with SQL server to let it integrate with the .NET framework [8].
4.3 ARM Cortex-M3 System Flow Charts

4.3.1 ADC Initialization

Start

ADC0 peripheral

ADC0

GPIO Port E is enabled for ADC0

Select ADC function for GPIO Pins

Enable Sample Sequence

Clear the Interrupt status flag

Stop
4.3.2 ADC Read Channels

Start

Trigger the ADC Conversion

Wait for Conversion

YES

Read ADC values

Store in a Buffer

Convert to string to display

Again store in a Buffer

Stop
4.3.3 Relays Initialization

Start

Configure Peripheral Enable GPIOs

Configure GPIOs PIN Type

GPIOs Pins Write to 0x00

End

4.3.4 Relays ON/OFF

Start

Write Value to GPIO Pins to SET

Write Value 0 to GPIO Pins to CLEAR

Read GPIOs Pin Status

End
4.3.5 ARM Cortex-M3 Microcontroller Logic

- **Start**
  - Initialize ADC & Read sensors analog data
  - Started conversion analog to digital form
  - Stored in buffer and frame an Ethernet packet
  - Initialize and started a LW TCP/IP server

- **Waiting for client request**
  - YES → Sends Ethernet packet to client
  - NO → Ready to accept controlling commands

- **Ready to accept controlling commands**
  - YES → Process the controlling commands
  - NO → Waiting for client request
4.3.6 Web server-PC Logic

Start

Create socket & send request data to a server running in the Microcontroller

After receiving data, split into appropriate manner

Stored in Database

Displays on Web page

Waiting for control requests

Frame a packet and sends to microcontroller for processing

NO

YES
4.4 Interlock Unit Mode of Operations

4.4.1 Manual mode

Start

Initialization of variables and sub functions

If Manual Mode

Auto reload mode

Scan the button and if any button press

If PDR heater Button pressed

If TX & PDR fan is good, then PDR heater is ON

If DR heater Button pressed (enable)

If TX & DR fan is good,
If HPA heater button pressed (enable)

- If PDR heater current sense is good, then PDR-anode supply is ON

- If PDR HT or Anode key pressed

  - If DR heater current sense is good, then DR-anode supply is ON

- If DR- HT Anode key pressed

  - If HPA heater current sense is good, then HPA anode supply is ON

A

If TX & HPA fan is good,

B

Continued…
If SSA is ON then only switch on RF

If RF button press (enable)

If PDR, DR, HPA heater and PDR, DR, HPA HTs sense signal are good then only switch on SSA

If SSA button pressed (enable)

If SSA is ON then only switch on RF
4.4.2 Auto Mode

Start

If PDR AF and TX are good

PDR heater is ON

If DR AF and TX are good

DR heater is ON

If HPA AF and TX are good

HPA heater is ON

If PDR heater is good

PDR-HT is ON

D

Continued…
If DR heater is good

If HPA heater is good

If PDR, DR, HPA heaters and HTs are enabled after time delay (good)

If SSA is good (ON)

Return to main
4.5 Software Development Tools

The Keil MDK-ARM, Microsoft Visual Studio 2010 and Microsoft SQL Server 2008 software tools are used to develop the system. There are various cross compilers available in the present embedded industry market like Keil MDK-ARM IDE, MPLab IDE and CCS (Code Composer Studio) IDE, Silicon-Laboratory IDE etc. These development tools facilitate users to efficiently/flexibly develop and debug application code. The ANSI C compiler and standard libraries are altered or enhanced to address the particulars of embedded target processors.

4.5.1 ARM Development tools

Embedded development tools and middleware for the wide range of ARM, Cortex-M, and Cortex-R based microcontroller devices are integrated into µVision which provides interfaces to ULINK and other third-party debugger adapters.

Keil MDK-ARM: Microcontroller development kit is the complete software development environment for ARM7, ARM9, Cortex -M, and Cortex-R4 processor based devices. MDK-ARM is specifically designed for microcontroller applications. It is easy to learn and use, yet powerful enough for the most demanding embedded applications. It provides the complete solution for software development of embedded applications for ARM processor based microcontrollers. It combines the ARM C/C++ compiler, the µVision IDE, and the RTX Real-Time Operating System, as well as middleware libraries. Together with ULINK debug adapters and evaluation boards, MDK-ARM provides a powerful development platform.

The ARM C/C++ compiler has been the reference compiler for the ARM architecture since 1990. It is continually developed and optimized, together with the
ARM processors, to combine high performance with small code size and a compact C library for embedded systems.

The Keil μVision is an integrated development environment with a project manager, editor, and debugger. The integrated device database configures the compiler, linker, debugging and programming parameters. The MDK-ARM professional edition adds middleware libraries for file systems, graphical displays and communication protocols for TCP/IP, USB, and CAN. MDK-ARM is CMSIS compliant.

The ULINK Debug and Trace Adapter family enables fast flash programming and debugging of target hardware via the USB interface. The ULINKpro adds streaming trace for Cortex-M series targets and enables features such as performance analysis and code coverage. Keil software designs and manufactures evaluation boards and starter kits to evaluate new microcontrollers.

![ARM software development tool modules](image)

**Figure 4.2:** ARM software development tool modules
The following are the features of Keil-4 Integrated development environment tool:

- Complete support for Cortex-M, Cortex-R4, ARM7, and ARM9 devices
- Industry leading ARM C/C++ Compilation Toolchain
- μVision4 IDE, debugger, and simulation environment
- Keil RTX deterministic, small footprint real-time operating system (with source code)
- TCP/IP Networking Suite offers multiple protocols and various applications
- USB Device and USB Host stacks are provided with standard driver classes
- Complete GUI Library for embedded systems with graphical user interfaces
- ULINKpro enables on-the-fly analysis of running applications and records every executed Cortex-M instruction
- Complete Code Coverage information about program's execution
- Execution Profiler and Performance Analyzer enable program optimization
- Numerous example projects help quickly become familiar with MDK-ARM's powerful, built-in features
- CMSIS Cortex Microcontroller Software Interface Standard compliant

The UV4 projects contain the all necessary GUI (graphical user interface) and debugger settings for any particular application. Typically one directory will contain the Keil project file, various other internal files used by the IDE, and the user source files. The IDE contains a source file navigator which shows all the files used in the project divided in order to file groups. The file groups need not be used (i.e., all code
can be in a single group) but provide a way to modularize code and select compile options for a subset of the project files.

‘μVision’ IDE incorporates a device database of supported ARM microcontrollers. In μVision projects, the required options are set automatically when selecting the device from the device database. μVision displays only those options that are relevant to the selected devices. The flexible window management system enables the drag and drop of individual windows anywhere on the visual surface. This interface allows to make better use of screen space and to organize multiple windows efficiently. The editor provides an optimized workflow with intuitive toolbars providing quick access to editor functions. Editor functions are also available in the debug mode allowing easy source code navigation and editing. The integrated source browser provides access to all application symbols, together with name, type, and class information. The browser allows to instantly navigating to the definition and references of any symbol. The μVision debugger can be configured as a target debugger or as a simulator. It provides a single environment to test, verify, and optimize the application. The debugger also simulates many ARM MCUs including their instruction set and on-chip peripherals.

The debugger provides windows and dialogs to monitor and control of system, these include

- **Memory Window** - review and modify the memory contents.
- **Watch Window** - view and modify program variables and lists the current function call nesting.
- **Symbol Window** - view debug symbol information of the application program.
- **Disassembly Window** - synchronized with the source windows making program debugging easier.

- **Register Window** - view and change register contents.

- **Call Stack Window** - view current call nesting including variable values.

- **Breakpoints Window** - define watch points and complex execution, access, and conditional breakpoints.

- **Browse Window** - search for objects in code.

The system viewer windows display peripheral registers that the processor can read and write to. They display the state, content, and name of peripheral registers. Content values are instantly updated by the target hardware as changes occur. Values can also be changed at run-time by typing a new value from within the system viewer window.

The advanced analysis tools work with the simulator or with target hardware via the ULINKpro streaming trace adapter. The configurable logic analyzer provides a graphical display of signals and variables. You may click on variable changes to display the instructions that caused change in the source code editor window. The debugger provides code coverage statistics to verify applications that require certification testing and validation. Color coding highlights the execution status of instructions helping to refine testing. The performance analyzer displays the execution time recorded for functions in application. Bar graphs display the time spent in a function, and the number of calls to it. The execution profiler records execution statistics for each CPU instruction, including the execution count and execution time for each instruction. These can be reviewed by the editor and dissembler windows.
**ULINK Debug Adapters:** Keil ULINK family of debug adapters connect the personal computer USB port to the target system (via JTAG or similar debug interface) and allows debugging embedded programs running on target hardware.

The all ULINK adapters enable the following features:

- Download programs to target hardware
- Examine memory and registers
- Single-step through programs and insert multiple breakpoints
- Run programs in real-time
- Program Flash Memory
- Connect using JTAG or Serial Wire modes
- On-the-fly debug of ARM Cortex-M based devices
- Examine Trace information from ARM Cortex-M3 and Cortex-M4 devices

The following are the types of ULINK Adapters:

**ULINKpro:** It delivers real-time data and instruction trace streaming via USB. ULINKpro supports, the Keil ULINKpro debug and trace unit to connect PC's USB port to target system (via a JTAG, Cortex Debug, or Cortex Debug+ETM connector). It allows to program, debug, and analyze applications using its unique streaming trace technology. The ULINKpro, together with MDK-ARM, provides extended on-the-fly debug capabilities for the Cortex-M devices. We can control the processor, set breakpoints, and read/write memory contents, all while the processor is running at full speed. High speed data and instruction trace are streamed directly to PC enabling to analyze detailed program behavior. It supports ARM7, ARM9, and Cortex-M devices.
The data and instruction trace for Cortex-M systems have a sophisticated application analysis including execution profiling and code coverage, using the high speed flash downloader to download the code and data.

**ULINK2:** Replaces the original ULINK adapter. The Keil ULINK2 debug adapter connects PC USB port to target system (via JTAG, SWD, or OCDS) and allows to program and debug embedded programs on target hardware. It supports various ARM7, ARM9, Cortex-M, 8051, and C166 devices

**ULINK-ME:** It offers a subset of the ULINK2 features. The new ULINK-ME debug adapter now supports the 10-pin Cortex debug connector available on the latest Keil Evaluation boards. The ULINK-ME connects PC USB port to target system (via JTAG or SWD) and allows to program and debug embedded programs on the target hardware. It supports ARM7, ARM9, and Cortex-M devices, delivered and supported only as part of Keil or OEM Starter Kits. It is a standard 20-pin (0.1") JTAG and 10-pin (0.05") Cortex debug connectors.

**Cortex-M Core Sight:** The core sight provides low cost trace capability for Cortex-M series microcontrollers. The ARM’s debug and trace architecture provides visibility and control of the entire system. It is used for hardware and software design and optimization. It supports modular, flexible architecture ARM and other IP, with established industry standards and comprehensive tool support. All Cortex-M based devices feature the ARM Core-Sight technology with advanced debug and trace capabilities. MDK-ARM, together with a ULINK adapter, uses these features to enable to debug program. The following features can be accomplished.

- Read/write memory and peripheral registers on-the-fly, while program is running at full-speed.
- Set up to 8 breakpoints while the processor is running.

- Control the CPU allowing program start/stop.

- Single step source or assembler lines.

All Cortex-M3 and Cortex-M4 devices provide data and event trace. MDK-ARM provides a number of ways to analyze this information while system is running:

- **Trace Window** - displays program flow by capturing timestamps, PC samples, and Read/Write accesses.

- **Debug Viewer** - displays the Instrumented Trace (ITM) output in a terminal window.

- **Exceptions window** - displays statistical information about program exceptions and interrupts.

- **Event Counters** - display real-time values of specific event counters providing performance indications.

- **Logic Analyzer** - graphically displays variable changes in captured data trace.

**Instruction Trace**: Cortex-M devices with embedded trace macrocell provide instruction trace. The Keil ULINKpro stream instruction traces directly to personal computer. This enables debugging of historical sequences, execution profiling, and code coverage analysis. The virtually unlimited stream of trace information enables MDK-ARM to provide complete code coverage of the program. Code coverage identifies every instruction that has been executed, ensuring thorough testing of our application. This is an essential requirement for complete software verification and certification.
Now-a-days, microcontroller applications often require simultaneous execution of multiple tasks in a real-time environment. While it is possible to implement an embedded program without using a real time kernel, the proven Keil RTX allows focusing on application development, enabling to save time, and produce a more reliable, expandable system. RTX is a royalty-free, real-time operating system specifically developed for the ARM and Cortex-M feature-sets. RTX source code is provided in all MDK-ARM Editions [9].

RTX provides features to manage system resources:

- Applications separated into independent tasks
- Extensive time control (scheduling, time delay/intervals)
- Deterministic execution times and task scheduling
- Inter-task communication, resource sharing, and memory allocation features with message pools
- Supports development with error checking, debug and test facilities

**RTOS Aware Debugging:** RTX is fully integrated in the µVision debugger making it easy to monitor task status and kernel activity. The RTOS-aware dialog is available in simulation and also when running on target hardware. It also displays information about all aspects of the kernel and the running tasks. This enables to view statistics about the active tasks, stack loading, and system resource usage. Microcontrollers offer a wide range of communication interfaces to meet any requirement in embedded design. However, implementing these interfaces presents software developers with real challenges. Middleware components are essential for developers to make efficient use of the device capabilities. MDK-Professional includes a number of royalty-free,
tightly coupled middleware libraries which enable developers to more easily implement complex communication interfaces in their applications.

- TCP Networking Suite
- USB Host and Device interfaces
- Flash File System
- CAN drivers

All middleware libraries have been specifically designed and optimized for ARM powered MCUs. The libraries are seamlessly integrated with the μVision environment and offer a modular design with well documented APIs.

**TCPnet Networking Suite:** The TCPnet library is a full networking suite optimized for ARM and Cortex-M processor based MCUs. It has a small code footprint, and delivers excellent performance. The TCPnet provides comprehensive support for transmission protocols such as TCP/IP and UDP, as well as application level services and clients including HTTP, Telnet, SMTP, SNMP, and FTP. It provides all the features required for modern networking communication in embedded systems [10].

**USB Interface (RL-USB):** Today the Universal Serial Bus (USB) is the standard way to connect external peripherals to a personal computer. Consequently, when designing an embedded system that has to interact with a personal computer, customers expect it to use a USB port. RL-USB is an easy-to-use USB software stack that provides a common API across a range of USB peripherals found on different microcontroller devices.

RL-USB describes the RL-USB Library designed to create the USB device and USB host applications. RL-USB is integrated in the Real-Time Library (RL-ARM).
The RL-USB library offers configurable functions to quickly design an application for a USB device or USB host. The library handles the low-level USB requests without the need to write the hardware layer code. Developers can focus on the applications request rather than concentrating on the specialties of the USB protocol. The USB host library is an embedded USB stack supporting USB MSC (Mass Storage Class) and HID (Human Interface Device) classes. It has been designed to be high performed while using as little memory as possible. The USB device interface uses standard device driver classes that are available with all Windows PCs. No Windows host driver development is required. The USB device interface uses a generic software layer using RTX kernel features.

**Flash File System:** The file system can be used to store program data during deep power saving modes, or for holding program constants, or even for storing firmware upgrades for a boot-loader. In short, a file system is a new and extremely useful tool for developers of small, embedded systems. RL-Flash File System (RL-FlashFS) is a software library that provides a common API to create, save, read, and modify files on a flash device. The library offers interface functions and handles the low level file input and output operations. The developer can focus on the application needs rather than concerning about the implemented file system. RL-FlashFS works with several ARM and Cortex-M processor based devices, and can be used standalone or with the RTX RTOS. RL-FlashFS applications are written using standard C constructs and are compiled with the ARM RealView Compiler. RL-FlashFS supports several media types, such as standard Secure Digital (SD), Secure Digital High Capacity (SDHC), Multi Media Card (MMC), and Flash Memory Cards. Flash File System Features lists the RL-FlashFS characteristics.
4.5.2 Microsoft Visual Studio

The Microsoft Visual Studio 2010 simplifies the entire C# code development process from design to deployment. The prototyping, modeling, and design tools help unleash creativity and bring vision to life. Coding is faster than ever, targeting different platforms, with integrated testing and debugging tools. One will find and fix bugs quickly to make sure the solution is at the highest possible quality [11]. It is an integrated development environment (IDE); a set of tools in a single application that helps write programs.

Visual Studio (VS) includes a suite of project types that one can choose from to start a new project. The VS will automatically generate skeleton code that can compile and run immediately. Each of the project types has project items that we can add, and project items include the skeleton code. Visual Studio offers many premade controls, which include skeleton code, saving one from having to write own code for repetitive tasks. Many of the more complex controls contain the wizard, which helps to customize the control’s behavior, generating code based on wizard options we choose. The VS editor optimizes the coding experience. Visual Studio introduces features, such as a call hierarchy, which lets see the call paths in that code; snippets, which allow us to type an abbreviation that expands to a code template; and action lists for automatically generating new code.

It has the toolbox jam-packed with controls, a server explorer for working with operating system services and databases, a solution explorer for working with projects, compilers, testing utilities, and visual designers.
4.6 Programming Languages

4.6.1 Embedded C

Microcontroller specific assembly programming language is reduced and embedded systems moved onto C as the embedded programming language. C is the most widely used programming language for embedded controllers. Assembly language is also used but mainly to implement the portions of the code with very high timing accuracy, code size, efficiency, etc. as prime requirements.

Initially, C was developed to fit into the space of 8K and to write portable operating systems. Originally, it was implemented on UNIX operating systems. As it was intended for operating systems development, it can manipulate memory addresses and allowed programmers to write very compact codes.

As assembly language programs are specific to a processor, assembly language didn’t offer portability across systems. To overcome this disadvantage, the high level language C came up. C got wide acceptance for not only embedded systems, but also for desktop applications. It still has a stronghold in embedded programming. Due to the wide acceptance of C in the embedded systems, various kinds of support tools like compilers and cross compilers, ICE, etc., came up and all this facilitated development of embedded systems using C language.

Key characteristics of an embedded system, when compared to PCs, are as follows:

- Embedded devices have resource constraints (limited ROM, limited RAM, limited stack space, less processing power)
- Embedded systems typically use smaller and less power consuming components. Embedded systems are more tied to the hardware.

The two salient features of embedded programming are code speed and code size. Code speed is governed by the processing power, timing constraints whereas code size is governed by available program memory and use of programming language. Goal of embedded system programming is to get maximum features in minimum space and minimum time.

Use of C in embedded systems is driven with its small and reasonably simpler to learn, understand, program and debug features. C compilers are available for almost all embedded devices. C has an advantage of processor-independence and is not specific to any particular microprocessor/ microcontroller or any system. As C combines functionality of assembly language and features of high level languages, it is treated as a ‘middle-level computer language’ or ‘high level assembly language’. It is fairly efficient, supports access to I/O and provides ease of management of large embedded projects. Offers more flexibility because C is relatively small, structured language; it supports low-level bit-wise data manipulation.

Embedded C is for microcontroller based applications, has to use with the limited resources (RAM, ROM, I/Os) on an embedded processor. Thus, program code must fit into the available program memory. If code exceeds the limit, the system is likely to crash. Compilers for ANSI C typically generate OS dependant executables. Embedded C requires compilers to create files to be downloaded to the microcontrollers or microprocessors where it needs to run. Embedded compilers give access to all resources which is not provided in compilers for desktop computer applications.
The programming with embedded C for embedded applications, we need to optimally use the resources, make the program code efficiency, and satisfy real time constraints. All done using the basic constructs, syntaxes, and function libraries of C. Embedded C uses most of the syntax and semantics of standard C, e.g., main() function, variable definition, data type declaration, conditional statements (if, switch-case), loops (while, for), functions, arrays and strings, structures and union, bit operations, macros, etc.

In addition, there are some specifics to embedded C which are mentioned below:

**Low Level Codes:** Embedded programming requires access to the underlying hardware, i.e., timers, memory, ports, etc. In addition, it is often needed to handle interrupts, manage job queues, etc. As C offers pointers and bit manipulation features, they are extensively used for direct hardware access.

**In-line Assembly Code:** For a particular embedded device, there may be instructions for which no equivalent C code is available. In such cases, inline assembly code, i.e., assembly code embedded within C programs is used; the syntax depends upon the compiler. The writing inline assembly code is much easier than writing fully fledged assembly code.

**Heap, recursion Features:** Embedded devices have no or limited heap area (where dynamic memory allocation takes place). Hence, embedded programs do not use standard C functions like malloc. Structures like linked lists/trees are implemented using static allocation only. Similarly, recursion is not supported by most embedded devices because of its inefficiency in terms of space and time. Such other costly
features of standard C which consume space and execution time are either not available or not recommended

I/O Registers: Microcontrollers typically have I/Os, ADCs, serial interfaces and other peripherals inbuilt into the chips. These are accessed as I/O Registers, i.e., to perform any operation on these peripherals, bits in these registers are read/write. Some embedded processors have separate I/O space for such registers. Since there are no such concepts in C, compilers provide special mechanisms to access. Such extensions are not a part of standard C, syntax and semantics differ in various embedded C compilers.

Memory Pointers: Some CPU architectures allow us to access I/O registers as memory addresses. This allows treating them just like any other memory pointers.

Bit Access: Embedded controllers frequently need bit operations as individual bits of I/O registers corresponding to the output pin of an I/O port. Standard C has quite powerful tools to do bitwise operations. However, care must be taken while using them in structures because the C standard doesn’t define the bitfield allocation order and C compilers may allocate bitfields either from left to right or from right to left.

Variable data types: Datatypes can be declared, and the compiler takes care of the storage allocation as well as that of code generation. But, datatypes usage should be carefully done to generate optimized code. Using long data types increase code size and execution time. The use of floating point variables is the intrinsic imprecise nature of floating point operations, alongside speed and code penalty.

Const and Volatile: Volatile is quite useful for embedded programming. It means that the value can change without the program touching it. Consequently, the compiler
cannot make any assumptions about its value. The optimizer must reload the variable every time it is used instead of holding a copy in a register. Const datatype is useful where something is not going to change, for e.g., function declarations, etc.

### 4.6.2 C# (C sharp)

C# is pronounced “see sharp” is an object-oriented programming language and part of the .NET family. The most recent version is C# 4.0 and it is part of Visual Studio 2010. C# is very similar to C++ and Java. C# works only on the Windows platform.

C# is a modern Object-oriented programming language. Object-oriented programming (OOP) is a programming paradigm using objects, data structures consisting of data fields and methods together with their interactions to design applications and computer programs. Programming techniques include features such as data abstraction, encapsulation, messaging, modularity, polymorphism, and inheritance [13]. C# has data types and variables, control flows (if-else) and loops (while, for loops).

The “Variables” are simply storage locations for data. One can place data into them and retrieve their contents as part of a C# expression. The interpretation of the data in a variable is controlled through “Types”. Boolean types are declared using the keyword “bool”. They have two values that are “true” or “false”. In other languages, such as C and C++, boolean conditions can be satisfied where 0 means false and anything else means true. However, in C# the only values that satisfy a boolean condition is true and false, which are official keywords. The special characters are that may be used in strings.
The *if* statement is probably the most used mechanism to control the flow in an application. It allows taking different paths of logic, depending on a given condition. When the condition evaluates to a boolean true, a block of code for that true condition will execute. Another form of selection statement is the switch statement, which executes a set of logic depending on the value of a given parameter. The types of the values a switch statement operates on can be booleans, enums, integral types, and strings.

The essential technique when writing software is looping the ability to repeat a block of code X times. The *while* loop is simply executes block of code as long as condition is true. A *do* loop is similar to the while loop, except that it checks its condition at the end of the loop. This means that the do loop is guaranteed to execute at least one time. A *for* loop works like a while loop, except that syntax of *for* loop includes initialization and condition modification.

In object oriented programming, objects are the basic building blocks of a program. Objects consist of data and methods. Methods change the state of the objects created. They are the dynamic part of the object; data is the static part. Methods are extremely useful because they allow separating logic into different units. One can pass information to methods, have it perform one or more statements, and retrieve a return value. Methods are similar to functions, procedure or subroutine. The difference is that a method is always a part of a class. The methods are two types that are static methods and non-static methods (instance method). Static methods are called without an instance of the object. To call a static method, we use the name of the class and the dot operator. Static methods can only work with static variables. Static methods are often used to represent data or calculations that do not change in
response to object state. We use the static keyword to declare a static method or a static variable. Non-static Methods belong to each instance created from the class.

Namespaces are used to organize code at the highest logical level. They classify and present programming elements that are exposed to other programs and applications. Within a namespace, we can declare another namespace, a class, an interface, a struct, an enum or a delegate. Namespaces prevent ambiguity and simplify references when using large groups of objects such as class libraries. Namespaces organize objects in an assembly it is a reusable, versionable and self-describing building block of a CLR application. Assemblies can contain multiple namespaces. Namespaces can contain other namespaces. An assembly provides a fundamental unit of physical code grouping. A namespace provides a fundamental unit of logical code grouping [14].

Everything in C# is based on classes. A class normally consists of methods, fields and properties. Every class has a constructor, which is called automatically any time an instance of a class is created. The purpose of constructors is to initialize class members when an instance of the class is created. Constructors do not have return values and always have the same name as the class.

**Constructor:** A constructor is a method in the class which gets executed when its object is created. Usually, we put the initialization code in the constructor. Writing a constructor in the class is damn simple. A class or struct may have multiple constructors that take different arguments. Constructors enable the programmer to set default values, limit instantiation, and write code that is flexible and easy to read.

Properties provide the opportunity to protect a field in a class by reading and writing to it through the property. In other languages, this is often accomplished by
programs implementing specialized getter and setter methods. C# properties enable this type of protection while also letting you access the property just like it was a field. The following are the object-oriented techniques:

**Inheritance:** Inheritance is one of the primary concepts of object-oriented programming. It allows you to reuse existing code. Through effective employment of reuse, you can save time in programming.

**Polymorphism:** Another primary concept of object-oriented programming is Polymorphism. It allows you to invoke derived class methods through a base class reference during run-time.

**Encapsulation:** Encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition. Typically, only the object's own methods can directly inspect or manipulate its fields.

In programming error and exception handling is very important. The C# has a built-in and ready to use mechanism to handle the exception. This mechanism is based on the keywords `try`, `catch`, `throw` and `finally`. Exceptions are unforeseen errors that happen in our programs. Most of the time, we can, and should, detect and handle program errors in our code. For example, validating user input, checking for null objects, and verifying the values returned from methods are what you expect, are all examples of good standard error handling that you should be doing all the time.

However, there are times when we don't know if an error will occur. For example, we can't predict when we'll receive a file I/O error, run out of system memory, or encounter a database error. These things are generally unlikely, but they could still happen and we want to be able to deal with them when they do occur. This
is where exception handling comes in. When exceptions occur, they are said to be “thrown”. A method will try to execute a piece of code. If the code detects a problem, it will throw an error indication, which code can catch, and no matter what happens, it finally executes a special code block at the end.

4.6.3 ASP .NET

"ASP.NET is a technology for building powerful, dynamic web applications and is part of the .NET framework". It is a development framework for building web pages and web sites with HTML, CSS, JavaScript and server scripting. It supports web pages, MVC (Model View Controller), and web forms.

Web page is one of the three programming models for creating ASP.NET web sites and web applications. Web pages are easy extendable with programmable web helpers, including database, video, graphics, social networking. Web page is the simplest programming model for developing ASP.NET web pages, and easy way to combine HTML, CSS, and JavaScript and server code. .NET is language independent, which means one can use any .NET supported language to make .NET applications. The most common languages for writing ASP.NET applications are C# and VB.NET.
REFERENCES


