**6.**

**Design and Development**

Embedded system, like any other product needs to go through several steps of development before it is finally ready to be used. Embedded system product development comprises of –

* Requirement Specifications
* Analysis
* System Design
* Hardware Software co-design
* Hardware firmware integration testing
* Implementation

After appropriate hardware components are selected and embedded hardware is designed and developed, then the firmware is designed.

In order to program the embedded system, a program is developed. A program is generally developed in assembly language, or higher level language. Assembly programs are either developed using hand assembly where hex code for mnemonics is manually looked up and fed to the microcontroller; or they are developed using a assembler where hex code is automatically fetched for the mnemonics. Higher level language programs on the other hand require IDE tools for development of program.

Overall, entire development environment has to be put together, to develop a good embedded system.

**6.1 The Embedded System Development Environment – IDE**

The development environment consists of development computer or host computer. This is the heart of development environment. An IDE tool is needed to actually develop the program. This tool is installed on the operating system of the host computer. A similar tool is needed for hardware design. It is known as electronic design automation (EDA) tool. Another requirement is for a hardware emulator tool for debugging the system. The tools are selected based on the target hardware development requirements.

A block dig of the development environment is shown in figure 6.1

In embedded development context, IDE stands for a target processor specific development and design environment. IDE package generally contains a text editor, used to edit the source code, cross compiler, linker and debugger. Some environments also have interface to hardware emulators, processor’s flash memory programmer etc. Figure 6.2 shows the constituents of embedded development environment.

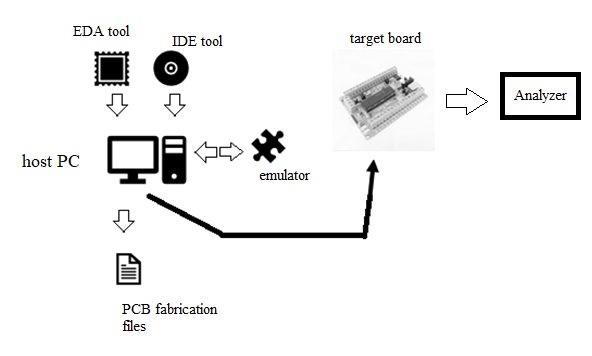
****

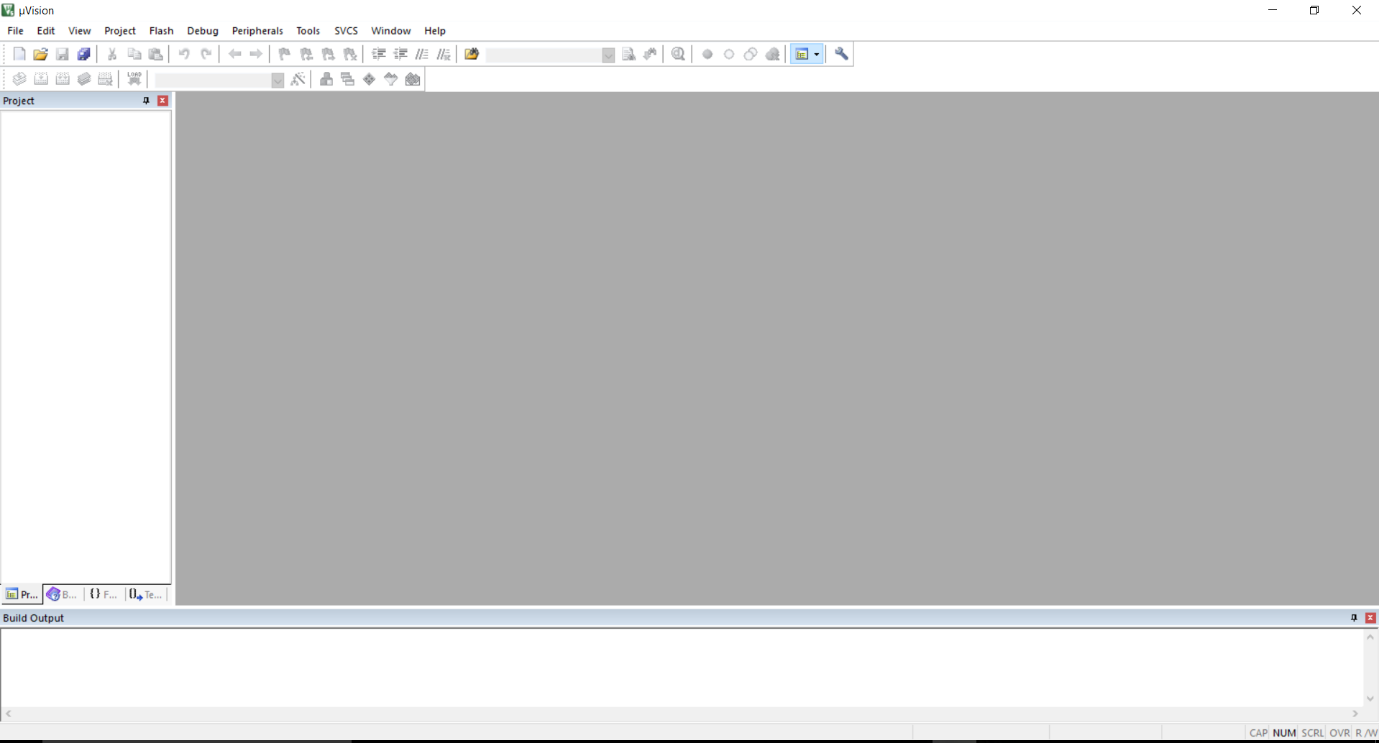
Fig 6.1 block diagram – embedded system development environment

Fig 6.2 constituents of embedded development environment

In embedded development applications each IDE is specific to controller family. A single IDE for all different microcontroller family is not yet available. However, Eclipse, an open source IDE is in development mode for the purpose of multiprocessor family development.

Example of a popular IDE development tool is µvision. µvision is a licensed IDE tool from Keil software ([www.keil.com](http://www.keil.com)) company. The demo tool can be installed from the web site. To start with the IDE, double click the executable file or choose the Keil µvision X (X is version number) from the start menu – all programs on the host computer with Microsoft © Windows operating system. The IDE view is shown in figure 6.3

IDE consists of project window consisting of Register view, menu bars, text editor window, output window etc.



Menu

Project window

Output Window

Text editor

Fig 6.3 Keil µvision 5 Integrated Development Environment

**Working with µvision**

Basic steps in using µvision 5 are given in chapter 5 in the example. They are discussed briefly as follows –

**Step 1**

Create a new project. Select the device for target. First select the vendor who manufactured the chip for the design. For example, use Atmel. Then chose the target processors exact number which will be displayed after selection of vendor. This is shown in figure 6.4.

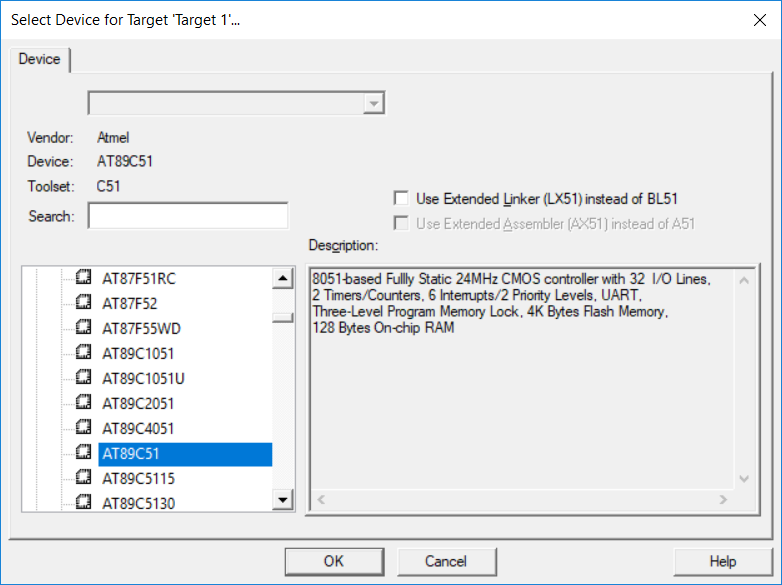


Fig 6.4 Target CPU Selection for Keil µvision 5 IDE

**Step 2**

Once target processor is selected, IDE will automatically add required startup code to the project. The dialog box shown in figure 6.5 is prompted.

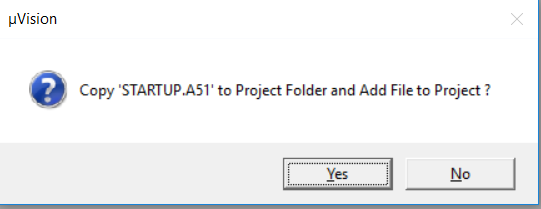


Figure 6.5 Startup File addition

After adding the startup files, cross compiler will add the code generated by startup files on top of the code generated for the main() function. The startup code contains default necessary initialization. For example, the stack pointer initialization.

A target group with the name Target1 is automatically created under Files section of the project window. Target1 contains the source group and startup file. This is shown in figure 6.6

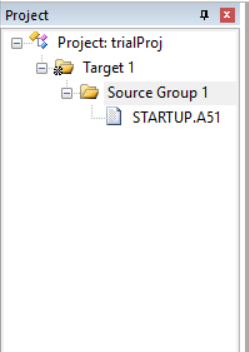


Fig 6.6 Project window with Target created and startup added

**Step 3**

Add a new file to the project. One method is to click on the File – New option. Now a text file will be created. It needs to be saved and then added to the project.

Another method will be to right click on the source group and directly add a new c file.

Now the file can be edited. A header file “reg51.h” is needed in the C program. This includes all target specific declarations for 8051 family.

The standard header files can be used. They have different implementation than the standard C implementations. For standard C implementation input output operations are performed with DOS console. However, for embedded C, input output operations are performed through sensors and actuator devices.

Figure 6.7 shows a C file added to the project to show the “hello world” message.

The programming implements a modular approach. Only one of the files in the source group can have the main() function. More than one file containing main() function will result into an error.

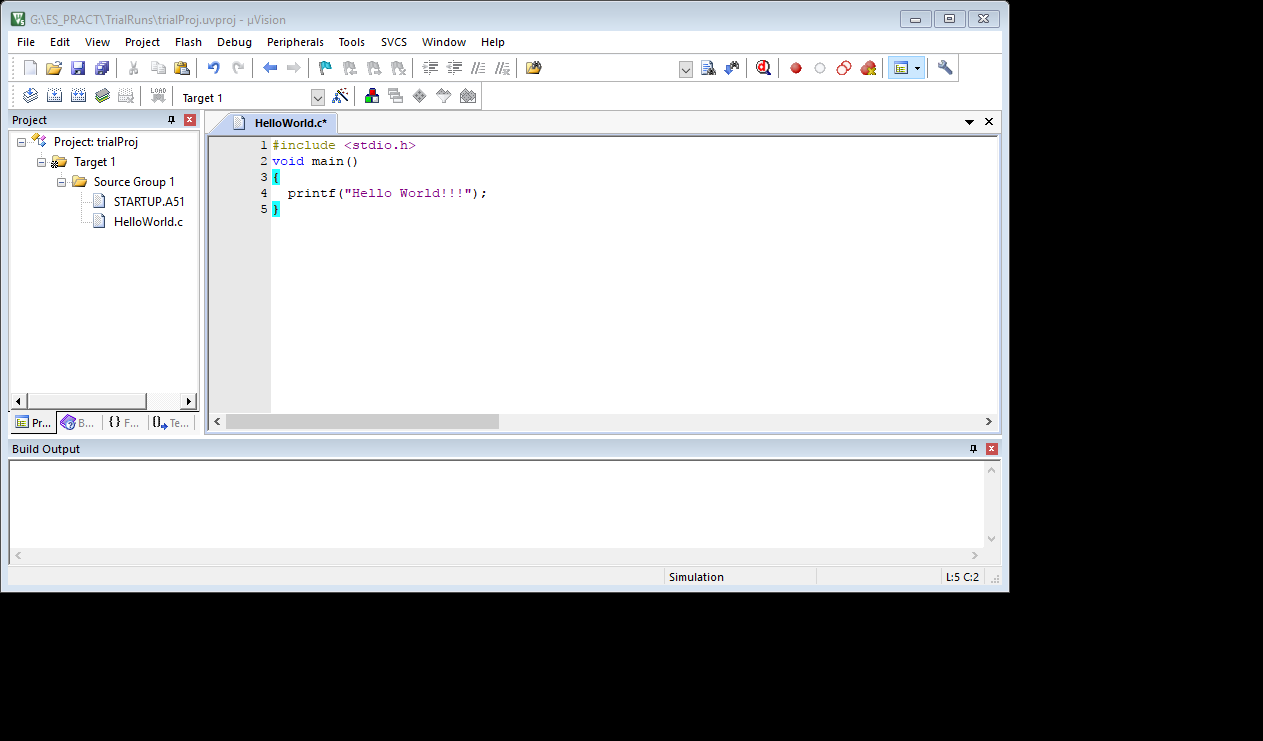


Fig 6.7 HellowWorld.c added to the project.

**Step 4**

Target must now be configured. For target configuration, go to the target properties by using the popup menu as shown in fig 6.8 from the options.

Device configuration is already done at the time of device selection. Now select the target tab and configure the following –

Choose on chip or external ROM. If external code memory is used specify the starting and ending address.

Select the clock frequency for the designed system. Typical values would be 6 MHz, 11.0592 MHz, 12 MHz etc.

Output tab holds setting for output file generation from the source code. the source file can either be converted into executable hex file, or a library file. The output and Target settings are shown in figure 6.9 and 6.10.

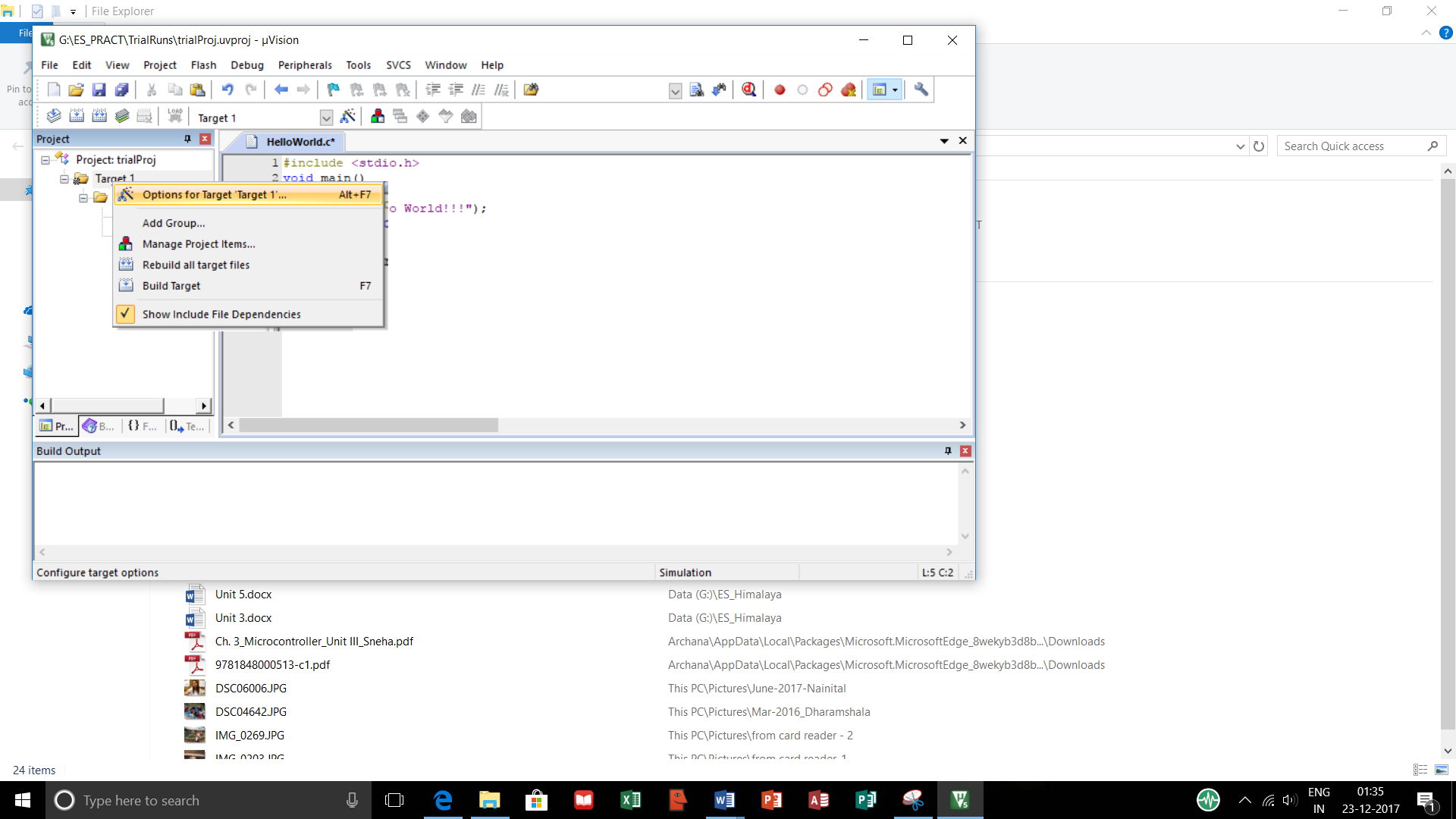


Figure 6.8 choosing options for target

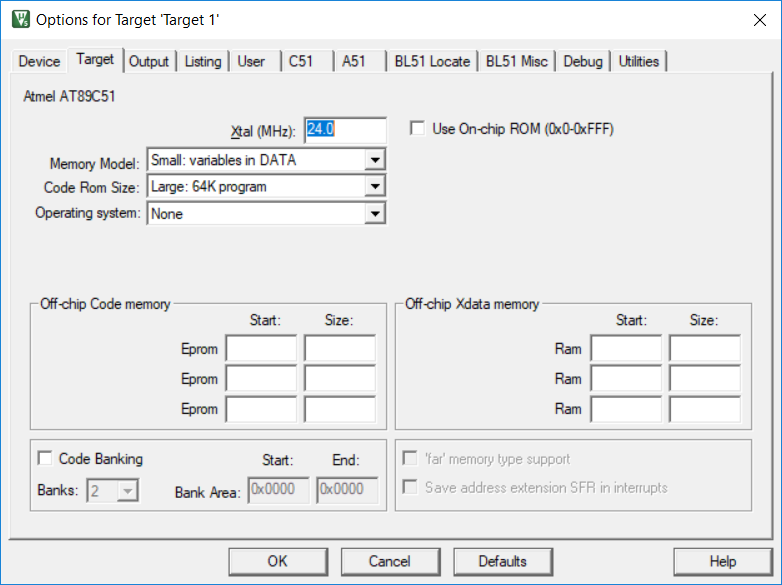


Figure 6.9 Target options

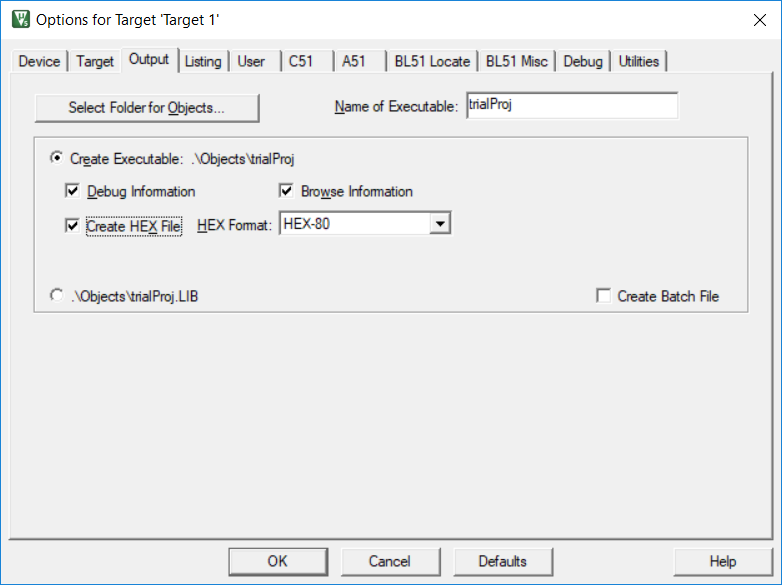


Figure 6.10 Output configuration

Debug tab consists of setting for firmware debugging. It supports both, simulation type firmware debugging and debugging for target hardware. If firmware option is selected, code may not be copied into target machine.

Hardware debugging is possible through the comm port to which target device is connected. This is shown in figure 6.11

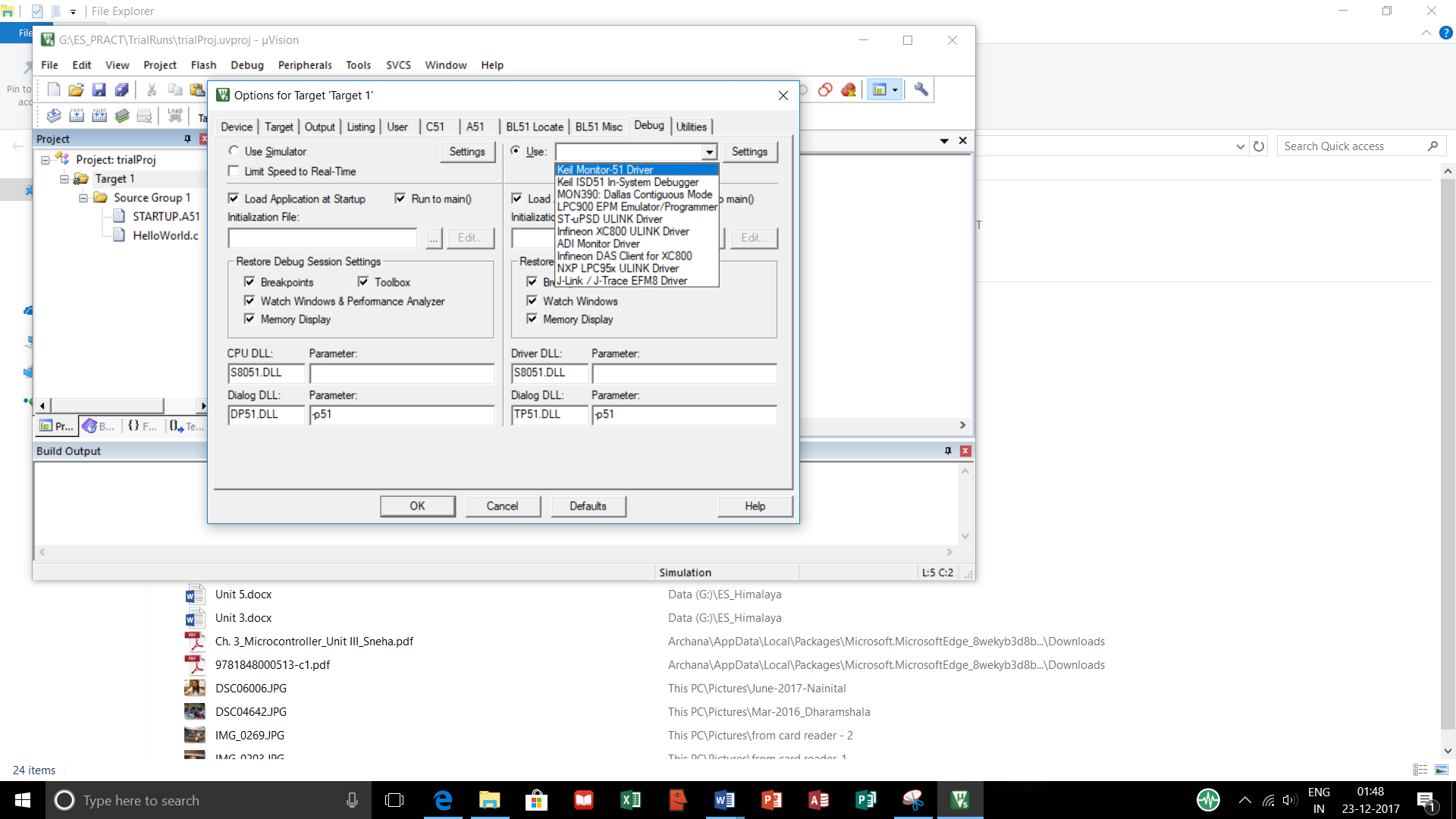


Figure 6.11 Firmware debugging options

**Step 4**

Compiling and building the target program can be done by using the build option. This will convert the code written in embedded C to machine language. The output window after building the target of the hello world program is shown in figure 6.12.

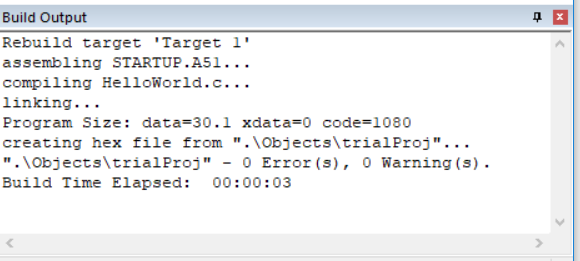


Figure 6.12 Build output for the HelloWorld program.

As seen in the output window, HelloWorld.c is compiled and linked using the startup.asm file for default initializations.

Output shows the size of data memory (data = 30), code memory (code = 1080)

The target is now ready. The IDE can be further used to simulate the output in the disassembly and register window.

**6.2 Types of files on cross compilation**

Cross compilation is the process of converting source code written in higher level language to target processor / controller understandable machine code. The conversion is done by a software tool (IDE) tool on the host computer. The software performing the task is actually the cross compiler.

If the source code is written in assembly language specific to the target processor / controller is written in assembly language, then the conversion is done by a program called cross assembler.

The task of cross compilers and cross assemblers is carried out through different steps and in the process generate different types of files as listed below –

* List files (.lst)
* Hex file (.hex)
* Pre-processor output file
* Map file (file extension depends upon the linker)
* Object file (.obj)

**List File**

Listing file is generated during cross compilation process. This contains lot of information regarding the cross compilation process. Following are the details about the listing file. They contain –

* Cross compiler details :

*Cross compiler details such as page header and command line used to invoke the compiler is listed in the lst file.*

* Formatted source text (C code)

*Source code listing has the line number and the source code written in that line. Cross compiler directives can be used to include or exclude the conditional statements from the source code. If properties are set as per the requirement, entire code of the header file(s) can also be included.*

* Assembly code generated

*Contains assembly code generated by cross compiler for the C code is included in the list file. Special directives can be used to not include the assembly code in lst file.*

* Symbol tables

*Information about various symbols present in the cross compiler’s C file. This listing contain following information about the symbols –*

*Name of symbol*

*Class*

*MSPACE (memory space)*

*TYPE (data type)*

*OFFSET (starting address of the code memory.*

*SIZE*

* Module information

*Provides size of initialised and uninitialized memory areas defined by source file.*

* Errors and warnings

This section contain the information about errors encountered during the execution. Levels of errors to be included in this file as per the settings done.

**Pre-processor output file**

The pre-processor output file generated during cross compilation usually contains the pre-processor output for the pre-processor instructions used in source file. They are mainly used for verifying the operation of the macros and conditional processor directives.

Pre-processor output file is always a valid C source file.

The extension is compiler specific.

**Object File**

Cross compiling / assembling converts various code files into object file. The format of the object file depends upon the cross compiler. C51 cross compilers support OMF51 format. Object file is a specially formatted file which has data records for symbolic information, object code, debugging information, library references etc. some of the details are listed below –

* Reserved Memory for global variables
* Public symbol names
* External symbol
* Library files
* Debugging information for synchronizing with object code.

The code does not have a permanent memory address. It is the responsibility of linker / locator to assign an absolute memory location in the code memory. Hence, the code generated by the cross compiler is not executable unless it is linked for resolving references to external memory.

**Map File**

If there are multiple source files in a project, each compiled source file will create a object file. The object files so created do not have fixed place in the code memory. They have to be linked to other object files and this task is done by a linker. Locator will then link the absolute address to each module in the code memory. This linking and locating generates a list file. This is known as the ‘linker list file’ or map file.

Map file contains information about link /locate process and is made up of separate sections. Different sections listed are –

* Page header

*Indicates linker version number, date, time and page number*

* Command line

*Represents entire command line that was used for invoking the linker.*

* CPU details

*Details about target CPU and memory model – internal, external and paged data memory.*

* Input Modules

*Names of all object modules, library files and modules included in the linking process.*

* Memory map

*Starting address, length, relocation type and name of each segment in the program.*

* Inter Module cross references

*Section name, memory type and the name of the modules in which it is defined and all modules in which it is accessed.*

* Program size

*The size of various memory areas as well as constant and code space for entire application.*

* Warning and errors

*Errors and warnings generated during linking program.*

**Hex File**

Hex file is the binary executable file created from the source code. The final or absolute object file is converted into processor understandable binary file. The utility used to do this job is known as object to hex file converter.

Hex file have different formats for different processor families. Intel Hex and Motorola Hex are two common formats used in embedded system applications.

* Intel hex

Intel hex format contains following fields –

* + :

*Indicates start of every Intel Hex file*

* + ll

*Record length field representing number of data bytes in the record*

* + aaaa

*Address field represents starting address for subsequent data.*

* + tt

*hex record type*

* + dd

*Data field representing one byte data.*

* + cc

*Checksum*

File format is shown in figure 6.13

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| : | Ll | aaaa | tt | dd | dd | ….. | cc |

Figure 6.13 Intel hex file format

* Motorola hex

Similar to Intel hex, Motorola hex is also an ASCII file where hex data is represented in ASCII format. Lines in Motorola hex file represent a record. Each record has hex numbers that represent machine language code. The general form of Motorola hex file is shown in figure 6.14. The fields of the format are listed below –

* + SOR

*Start of record. ASCII character ‘s’ is used.*

* + RT

*Record type. Different record types like header, data, end of data record can be identified by this field.*

* + Length (ll)

*Count of character pairs in record*

* + Start Address

*Access field representing starting address*

* + Code /Data (dd)

*Represents one byte of data bytes. No of record is given by ll field.*

* + Checksum

Represents checksum of the record.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| SOR | RT | Length | Start Address | Data / code | checksum |

Figure 6.14 Motorola hex Format

**6.3 Disassembler / Decompiler**

Assembler converts the assembly language program to machine code. Reverse of this operation is disassembly. Disassembler is a device that converts machine code to assembly.

Similarly, decompiler is a device that converts machine code into a higher level language program.

Disassembler / decompiler for different processor family are different. They are deployed in reverse engineering. Disassembler / decompiler help in reverse engineering by producing higher level language code by analysing embedded firmware.

Disassembler / decompiler also used for detecting malicious code in an executable file (antivirus).

It is unlikely that disassembler / decompiler will generate exact replica of the original code. as many symbolic constants and comments are used.

However, a matching source code can be generated by using disassembler / decompiler.

**6.4 Simulators**

Simulator is a software tool which is used to simulate various conditions for checking the functionality of the application firmware. The IDE provide simulator support and help in debugging the firmware to check the required functionality.

Simulators simulate the target hardware and firmware can be monitored using the simulators. Simulator based debugging provide following features –

* Software based
* No real target system needed
* Primitive
* Lack of real time behaviour

**Advantages of Simulator based debugging**

Simulator based debugging have following advantages –

* No need for target board

*Software support of the IDE simulates the CPU of the target hardware. Firmware should be written as per the memory model of the hardware. Actual hardware is not needed and hence, development can start very early, even before the hardware is built. Actual hardware design can be improvised*

* Simulate I / O peripherals

*As option of simulating I/O devices is available, the need for connecting I/O devices for is not there anymore.*

* Simulate abnormal conditions

As physical hardware is not needed, simulation can be done only for specific condition of input. This helps the debugger to study the condition of the firmware for abnormal conditions of data.

**Limitations of simulator based debugging**

* Deviation from real behaviour

*Sometimes, all possible combinations of input may not be simulated. In this situation, a particular combination of inputs may give a particular output, but actual result may be different*

* Lack of real timeliness

*Major disadvantage of simulator based debugging is that they can not be implemented in real time.*

**6.5 Emulators and Debuggers**

Debugging process is a process through which a system performance can be diagnosed. In embedded system, a debugger would monitor registers of the embedded processors, memory elements, and system buses when the firmware is in execution. This helps in precise understanding the program efficiency and hardware functionality.

Embedded system debugging is classified into two categories

* **Hardware debugging**

Deals with monitoring system hardware like bus and status lines etc. This helps in finding and fixing an error or bug in hardware. Recently developed IDE tools have made the job quite easy. Earlier, every program had to be burned into EEPROM before the working of hardware could be seen. This involved production losses.

**Incremental EEPROM Burning technique for hardware debugging**

The most primitive method or the type of firmware debugging technique is “***Incremental EEPROM Burning Technique”.*** In this technique instead of burning the entire code on EEPROM, code is burned in incremental order. Each part of the code that has one function is separately coded, cross compiled and burned into the EEPROM. Each part of the code needs to have a indicator like LED as part of it, so that its working could be tested. If the code works properly, that is if the first functionality works, then the next is cross compiled and added to the system. After all functionalities are tested, then the software is put together and entire program is compiled and burned.

Although it is a time consuming process, it is effective and is used in many small scale embedded systems.

* **Firmware debugging**

Deals with monitoring of firmware execution. This helps in finding and fixing error or bug in the firmware program. These programs, with the help of the new tools, could be debugged step by step or batch by batch where a few instructions can be grouped together in a batch.

There are number of firmware debugging techniques used in embedded system development process. Some of the techniques are discussed below –

**Inline Breakpoint Based Firmware Debugging**

A primitive method in firmware debugging is ***“Inline Breakpoint Based Firmware Debugging”.*** In this technique inline breakup code used in the program. This code can be simple printf() statement which will show if the specific part of the code has executed as expected. This code is added immediately after the code segment to be debugged. Such debug codes do similar job as that of LED indicators in hardware. The data generated by printf() statements can be seen at the hyper terminal. Hyper terminal is a communication interface provided by the windows operating system. If host PC is connected to target hardware board with the help od RS232 on serial COM port, a hyper terminal can be configured to use the serial communication and output of printf statements can be viewed on hyper terminal. Figure 6.15 shows hyper terminal output of hello world program.

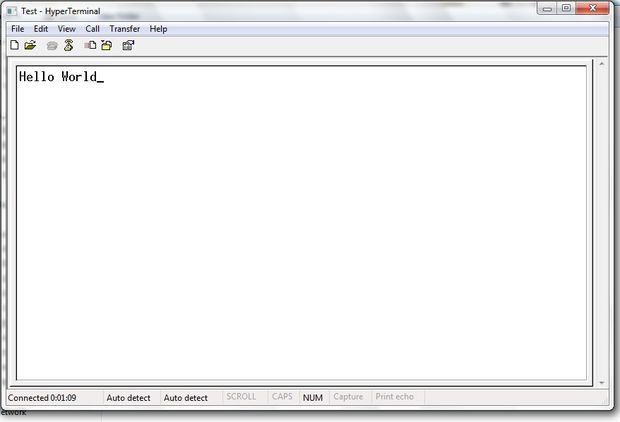


Figure 6.15 Hyper terminal output for the Hello World Program.

**Monitor Program Based Firmware Debugging**

Another approach in firmware debugging is ***“Monitor Program Based Firmware Debugging”.***

In this approach a monitor program controls the downloading of user code into memory, inspects and modifies the contents of memory / registers, allows single step execution to check the outcomes of every instruction in the code. monitor program can always communicate with the serial port of the target device. And as per the commands received through the serial interface, it reads and executes these commands. After all commands are executed, monitor program development is complete. The program is then burned into the FLASH memory of the target board. The ROM or the FLASH containing the monitor program is called the ‘monitor ROM’.

* Following are basic features of monitor program –
* Command set interface, to communicate with running application.
* Firmware download option
* Examine and modify processor’s register and memory option
* Single step program execution
* Set breakpoint in execution
* Send debug information to the processes of the host machine.

**Drawbacks of monitor based firmware debugging –**

* A major drawback of monitor program based approach comes up with systems having core like 8051. The issue is the monitoring program uses the actual code memory in RAM with Von Neumann architecture. As this architecture assumes a common bus, it shares memory map between ROM, monitor program data memory, external user memory and all other memory elements. However, controller like 8051 uses Harvard architecture where communication over devices is done on individual buses. Hence, the limited 64k code memory of 8051 needs to accommodate all memory requirements of monitor program. This can affect the performance.
* As the monitor operations use serial interface, one serial port of the target board becomes dedicated port for the monitor program. It hence can not be used for any other operations if the target board needs it.

**Difference between simulator and emulator**

Simulator and emulator have the same basic functionality – debugging the target. However, they follow different implementation techniques.

Simulator is a software application which first replicates the target CPU. Then it simulates the features and functionalities provided by the target.

Emulator on the other hand, is a self contained hardware device which emulates the target device. Emulator hardware contains necessary emulation logic and is connected to the target board through some interface.

This is demonstrated in figure 6.16

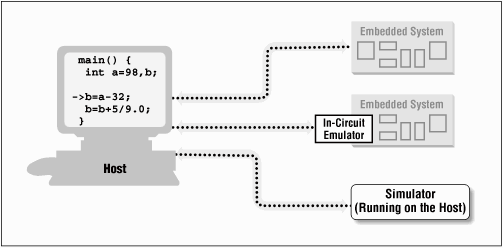


Figure 6.16 different implementations of simulator and emulator

**In Circuit Emulator Based Firmware Debugging**

With the advancements in embedded technology, emulators have evolved and can also operate like simulators. Now, they can be termed as software emulators. The debugger applications so built with software emulators can be integrated in IDE or a third party tool. An emulator POD is one such popular implementation. This in circuit emulator based system contains following functional units –

* Emulation Devices
* Emulation Memory
* Emulator Control Logic
* Device Adapters

**On Chip Firmware Debugging (OCD)**

As the semiconductor technology has advanced to a great extent, almost all processors / controllers incorporate built in on chip debug modules. The OCD adds complexity of the circuit and also increases the cost. However, it provides a great utility in debugging. On chip debug facility are vendor specific. Many OCD provide dedicated debug module with facilities of setting easy breakpoints and querying internal state of the chip and single step execution. Many vendors provide a ‘on chip debug support’ through a protocol called Joint Test Action Group (JTAG) protocol.

**6.6 Embedded System Product life-cycle**

**6.6.1 Introduction to Embedded Development Lifecycle**

Embedded product development lifecycle is a analysis – design – implementation approach for embedded system design. This approach is necessary to understand scope and complexity of the work involved in any embedded product development. In this approach different components in embedded product development, their correlation and interaction can be designed and implemented. This approach follows certain standards and these standards provide uniformity in development of various developers and vendors making embedded hardware and firmware.

Basic aim behind any embedded product development is to generate marginal benefit. In other words, to earn profit. A product is profitable if the sale cost of the product is more than that of overall investment and expenditure.

Following are the three ways in which profit can be ensured in embedded product development –

* **Ensuring High Quality Products**

When an embedded product is launched in commercial market, the current competition in the market must be analysed. The product should be having all the quality attributes provided by its competitors plus more. Quality attributes like response, throughput, speed etc. are main driving factors behind the success of embedded products.

* **Risk Minimization and Defect Prevention**

When embedded product development begins, it must be ensured at early stages of development that the product should not have major flaws. This ensures that the risk of launching such a high quality – defect free product in the market is minimum.

The task of risk minimization and defect prevention can be done by introducing project management process to the entire product development life cycle.

Project management makes important decisions in the development process more systematic. This includes cost estimation, resource management, time management.

* **Increased Productivity.**

Productivity is measure of efficiency. To increase productivity is to manage the available resources more efficiently. The resources include the available manpower, available budget, available time, available components etc.

To use the resource efficiently, reuse of resources is an important factor. The resources including components and code modules should be designed in such a way that reuse is possible.

In hardware design of embedded products, easily available (COTS) components are essential in the design. These components are easily available, low cost and maintainable.

Specially skilled developers save time of development as they have expertise in particular development technology.

**6.6.2 Different Phases of Embedded Product Development Lifecycle**

Different phases of embedded product lifecycle are more or less like any other product development phases. They are shown in fig 6.17 -

Fig 6.17 Different Phases in Embedded Product Development Lifecycle.

* **Requirement or Need**

In embedded industry, often need of a particular product arises. The need may arise in one of the following ways

* + New or Customized Product Development

A completely new product, which is not available in the market comes under this category. Some technical advances in other products or simply the taste of users may lead to the need of a new product altogether. For example a gaming console.

* + Product Re Engineering

Sometimes existing products need to be redone or re engineered. Again, reason for the need to re-engineer a product can be found in advancements in technology. For example, power windows or power steering in place of original manual arrangements.

Change in business requirements or need to have a better user communication interface can also lead to need of re-engineering of a product.

* + Product Maintenance

There are two ways of dealing with the need of product maintenance. First to repair and keep using the product. This is necessary when the product is new in the market and is having any teething issue.

Second way is to perform a preventive maintenance. This is done when the product has served its time in the market and is ready to retire. The users are facilitated to change to a new product.

* **Conceptualization**

Concept development phase for a product begins immediately after a concept proposal is prepared by the development team, is reviewed by the senior management and is formally approved. This phase defines scope of the concept, performs cost benefit analysis. It then performs feasibility study and prepares project management and risk management plans.

Feasibility study is performed with respect to technical as well as financial feasibility of the product.

* **Analysis**

Once the documents conceptualization documents are formally accepted by the client or sponsor of the project, then the analysis phase begins.

In this phase, the product is defined in detail with respect to the input, processes and outputs at the functional level. It determines the functions performed by the product.

Various activities performed in this phase are –

* + Analysis and Documentation.

*All functional entities like performance, quality attributes are listed in the analysis document.*

* + Interface Definition and Documentation.

*Embedded product can function individually, or as a member of a large distributed system. If it is part of a system; it will need an internal interface between the components. It will also need external communication interface to communicate with various input output devices. The list of device interfaces needs to be part of the document.*

* + Defining test plan and procedures.

*Different tests to be performed should be planned and designed prior to actual development. A master test plan is needed which will cover all functional as well as non functional aspects.*

*Different types of testing performed in product development life cycle are –*

* + - *Unit testing*
    - *Integration testing*
    - *System testing*
      * *Usability test*
      * *Load test*
      * *Security test*
      * *Scalability test*
      * *Smoke test*
      * *Performance test*
      * *Endurance test*
    - *User acceptance test*
* **Design**

Once the analysis is done, next step is to design the product. Product design starts with high level design. This design establishes top level architecture for the product, lists the various functional blocks. The design then lists the input and output for the each functional block.

After this design is ready, next step is to make the detailed design. It illustrates the detailed architecture and identifies components for each functional block.

Traditional embedded product designs were divided into hardware and software design. However, the modern design is the system design which includes hardware, software / firmware design.

The design documents are then prepared. These include –

Preliminary design document

Detailed design document

Operation Design document

Maintenance Design document

Product Training Design document

* **Development & Testing**

Development phase is when the product in plans is physically realized. Here, the detailed specifications generated during design phase are translated into hardware and firmware.

First step is to install and set up various development tools. Embedded hardware development is done using the CAD / CAM tools.

The next step is to procure the components. The components need lead time as some time is required for the delivery of hardware components.

Finally, the components are built together.

Once all components are built together, they need to be tested. Testing is pre planed in the analysis phase.

Testing phase can be independently implemented for firmware and hardware. This is the unit testing phase.

After the hardware and firmware integration takes place, the product is again tested. This is the integration testing phase.

Once the system is ready to use together, after the integration testing, its functionality is tested for usability, security and load. The overall performance is tested. System is also tested for endurance. This is the system testing phase.

All the test reports are documented as they will be useful if the next version of the product is to be developed.

* **Deployment**

This is the process of launching the first model of fully functional model of the product in the market; or handing over the product to the client, if it is a custom made one. During this phase product modifications as per the various integration tests are implemented and product is made fully functional.

Product launch notifications are sent to all stakeholders of the product. Notifications regarding –

Deployment schedule

Product description

End users

Features supported

Product support information.

Once the notification schedule is in place, there must be a training plan for the product. Before the actual launch of the product, end users must get the training to use the product.

Installation of the product will be the next step in deployment. The product installation can be a single step operation, or all components need to be separately installed.

Finally, post implementation reviews are taken from the users.

* **Support**

Support phase deals with maintenance of product in production environment. After the product is launched, it must be scrutinised to ensured there is no unexpected malfunctioning of the product, or operational error.

Support is provided to the end users to fix any bugs. Many times separate support teams work 24 x 7 to ensure that the end users can smoothly use the product.

In addition to identifying the bugs, support phase also identifies areas of improvements of the product.

* **Upgrades**

In this phase, new upgraded versions for the product which is already present in the market. During the upgrade phase, major bugs are fixed and improvements are implemented. Embedded products, upgrades can be upgrades in hardware or firmware. In case if any upgrades are required in firmware. This is known as firmware upgradation. This can be very easily done.

Sometimes hardware upgradations are needed. They can be brought about as hardware upgradation in the next version of the embedded product.

* **Retirement**

Any product can not sustain infinitely in the market. After a certain time, the product gradually starts wearing out of the market. Product retirement is a gradual process and it has several reasons line advancement in technology, enhanced user needs and expectations, or a better competitive product in the market.

Life time of the product in the market can be different for different products. For example Nokia handset 3310 was launched in 2000 and was finally discontinued in 2005 giving it a almost 5 year lifetime.

**6.6.3 Modelling the Embedded Product Lifecycle**

Modelling refers to the way in which interconnection of various phases involved in development of embedded product. Different approaches or models adapted in embedded product lifecycle are discussed below –

* **Linear or Waterfall Model**

Waterfall model is the classic model of product development. As per this model all phases of product development are executed in sequence. It establishes a formal analysis and design methodology with highly structured development phases.

The flow of execution is unidirectional. The output of one phase serves as input to the next phase. Feedback of each phase can be considered only after execution of the phase is complete. There is an extensive review considered after all phases are executed.

However, if there is a bug in a particular phase, it can not be fixed until all phases are executed.

The major advantage of this model is the product has a rich and elaborate documentation. The project management is easy and has a good control over cost and schedule.

Major drawback is it assumes that everything is in correct order and does not allow any changes during any phase of development.

Thus, this model is best suited for the products where the requirements are well defined.

Figure 6.18 shows the linear / waterfall model for embedded product development.

Conceptualization

Design

Analysis

Need / Requirement

Retirement

Upgrades

Support

Deployment

Development and Testing

Figure 6.17 WaterFall Model

* **Iterative / Incremental or fountain model**

Iterative or incremental model follows another sequence of execution. This uses an incremental approach. That is the bugs and shortcomings of certain stage can be improved before the next stage of implementation is executed. Thus, this can be considered as series of waterfall models, or cascaded series of waterfall models to be precise.

Changes in functionalities can easily implemented during execution.

Advantages of this model are –

Easy project management

Product development can stop at any stage with minimum risk.

Disadvantages –

Extensive review requirement at each cycle.

Training at every cycle.

This model is used where there is a high amount of risk. If waterfall model is used, the entire risk is at stake where as in iterative model, risk is divided over cycles.

* **Prototyping / Evolutionary Model**

This model is similar to iterative model and the product is developed in multiple cycles. The difference between the two is, this model produces a more refined prototype of the product at the end of every cycle. The commercial development of the product will be done one time and not at the end of every cycle. The problems and shortcoming of the prototype are however, evaluated at the end of every cycle and fixed in the next cycle.

There could be drawbacks like –

Deviation from the estimated cost.

Increased project management

Minimal documentation

Increased configuration management activities.

Figure 6.18 shows the prototype model –

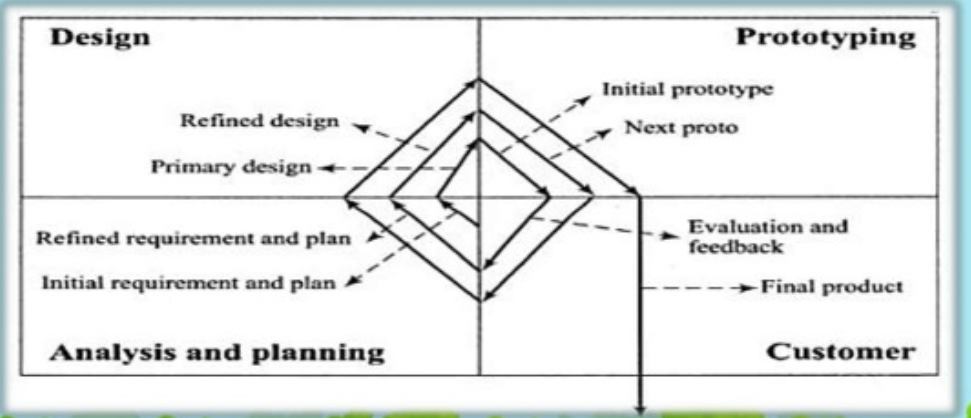


Figure 6.18 Prototype Model

* **Spiral Model**

Spiral model combines the best features of linear and prototype models. This model was developed by Barry Boehm in 1980s. As per this model, product development starts with project definition and then goes through all phases of product development multiple times. All the activities are distributed in four quadrants of a spiral. They are as follows –

1. Determine objectives, alternatives and constraints.
2. Evaluate alternatives. Identify and resolve risks.
3. Develop and test
4. Plan

Figure 6.19 shows the spiral model.

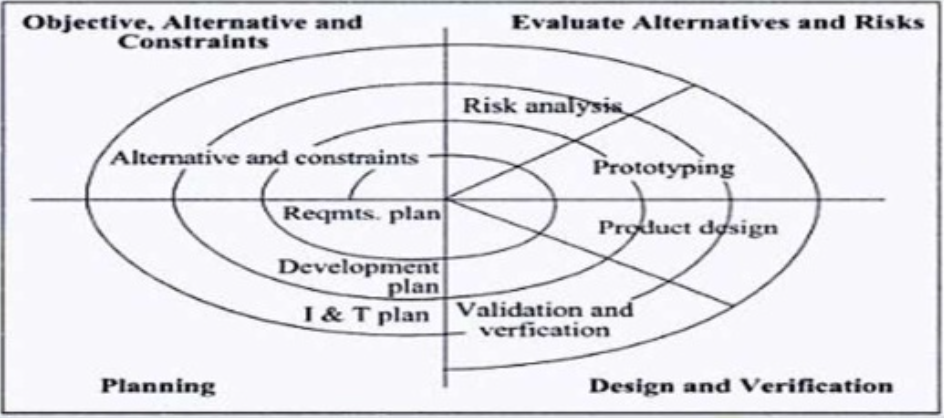


Figure 6.19 Spiral Model

Spiral model is used in development of complex embedded products. And, also when requirements are changing from customer side.

**6.7 Trends in Embedded Industry**

Embedded systems have evolved tremendously since their first mass production in 1990s. current trends and bottlenecks in embedded industry are discussed below –

**6.7.1 Processor Trends**

Most important change bringers to the embedded industry are processors. Processors and controllers have been through revolutionary changes since they were first introduced in 1970s. Embedded systems started with use of 8 bit processors. They are still in use for small scale systems. However, todays 8 bit processors are more advanced, power and feature rich.

Also, with high degree integration, and IP Core reuse techniques, functionalities of multiple ICs are integrated on single chip. This is known as system on chip (SoC).

Embedded industry has also witnessed architectural changes in the processors. RISC and instruction pipelines contributed greatly in improving the performance of the microprocessors.

Processor clocks are now giving clock frequencies in the order of few GHz as against the very low frequencies in initial stages. This has increased their power consumption.

There are following categories of advanced embedded processors

* **SoC (system on chip)**

*Multiple systems are integrated on single chip. They are used in diverse application needs like set top boxes.*

* **Multicore Processors / Chiplevel multi processor (CMP)**

*Incorporate multiple cores on the same chip and work on the same clock frequency supplied to single processor core. This prevents from having very high power consumption. Each core of CMP implements pipelining, multithreading etc. and enhances the performance.*

* **Reconfigurable Processors**

*Reconfigurable processors are the ones with the reconfigurable hardware features. They contain array of programming elements. Hardware features can be changed statically or dynamically. This brings a great advantage as it makes the chip adaptable to the firmware.*

**6.7.2 Embedded OS Trends**

Trends in processors are better utilized by advances in operating systems. A good number of commercial and open source operating systems are available to select from. Most of these new embedded operating system try to adapt the concept of virtualization by implementing microkernel architecture. The kernel contains only selected and most important features of operating system.

New trends in development also provide of the shelf device specific implementation. For example, Microsoft operating system have devised versions specific to multimedia embedded devices like media player.

Ubuntu MID edition of embedded linux designed for mobile internet devices is another example of this.

**6.7.3 Development Languages**

Embedded system development is done in following two aspects –

* System side application (embedded firmware)
* User application development (embedded software)

Embedded firmware is embedded in the hardware device’s program memory and it can not be altered by the end user. However, embedded hardware can be embedded in hardware, or user can install the software later. For example, a PDA.

First and foremost and most widely used development languages for embedded firmware development is Embedded C. it was popular because of the flexibility provided by C language and the capacity of it to access the hardware.

**JAVA as embedded development language**

With the changing trends, object oriented languages like C++ and JAVA are also used in embedded programming. JAVA is not very popular embedded development language as it has a shortcoming in supporting real life systems. As java code is interpreted by the JVM, it abstracts the processor dependency. Thus making the hardware implementation slower. A new java technique called JIT (just in time) compiler, speeds up the execution.

JAVA as embedded development language is not very popular and has following limitations

* Interpreted JAVA version is slower and does not meet real time requirements of embedded systems
* Garbage collector of JAVA is non deterministic in executional behaviour and unacceptable for embedded systems
* Processors which do not have built in JVM support require ported JVM to processor architecture.
* JAVA supports limited hardware resource access.
* Run time memory requirement of JAVA is high.

Some steps are taken to overcome the above limitations. They are listed below -

* A new Ahead of Time (AOT) compiler of java is launched which will convert bytecode into processor specific assembly code.
* JNI (Java Native Interface) libraries provide access to functions written in other languages like C
* A real time version of JVM is also developed
* Java Platform Micro Edition (J2ME) is customised version for specific design for embedded application software.

**.NET CF for embedded development**

.NET framework, a pre coded library collection acts as the run time component of .NET supported languages like C++, C#, J# and VB. It contains class libraries for user interface, database connectivity, network connectivity, image editing and so on. The runtime environment is called as Common Language Runtime. It also provides features like cryptography, garbage collection etc.

.NET is targeted for embedded system development for desktop applications. This is helping embedded software designs to be feature rich.

**6.7.4 Open standards and Frameworks and alliances**

Certain areas in embedded system development are popular and competitive. They bring in different combination and alliances through R & D. this helps the hardware manufacturers to come up with new designs and software manufacturers to develop supporting software.

Some of the popular alliances are –

* Open mobile alliance (OMA)

*Develops alliance for mobile phones.*

* Open Handset Alliance (OHA)

*Business alliance formed by various handset manufacturers. (HTC, Samsung, LG, Motorola etc.)*

* Android

*Open source software platform and operating system for mobile devices. Developed by google corporation and retained by OHA. Based on the Linux kernel, it supports a set of JAVA libraries. These libraries can be used for android application development.*

* Openmoko

It a project for building open hardware and firmware standards for a family of open source mobile devices. The target operating system is Openmoko Linux, which is a embedded Linux system.

**6.7.5 Bottlenecks**

* **Memory performance**

*Although processor speed for different operations has advanced to a great extent due to developments in technology, memory performance does not often match up with this speed. This creates a bottleneck. Alternate memory technique is not yet available to improve the overall performance.*

* **Lack of Standards**

*There are no specific standards in place to ensure interoperability in all areas of embedded system development. Even if a few standards are in place, not all the stakeholders and manufacturers are following the standards. As a result, embedded products have different architecture and designs. To make these product better in design and function a combined effort with defined standard are required.*

* **Lack of skilled Resources**

*The most crucial issue in embedded industry is lack of skilled resource persons to perform the task of embedded system design. Lack of new workforce, lack of good books and resources in the subject is making innovation and development more difficult.*

**Summary**

Design and development of embedded system is a process which goes through following stages –

* Requirement Specifications
* Analysis
* System Design
* Hardware Software co-design
* Hardware firmware integration testing
* Implementation

Embedded system development environment is required to develop embedded system design.

Development environment consists of –

* Host computer
* Tools required for firmware development (IDE)
* Tools required for hardware development (EDA)
* Hardware emulator tools / target boards

One popularly used development environment IDE tool is Keil µvision 5.

The process of creating an absolute object file from higher level language code, such as embedded C code, is known as cross compilation. The program doing this task is cross compiler.

Different files generated in the process of cross compilation are –

* List files (.lst)
* Hex file (.hex)
* Pre-processor output file
* Map file (file extension depends upon the linker)
* Object file (.obj)

The process of creating an absolute object file from assembly language code is known as cross assembly. The program doing this task is cross assembler.

Simulator is a software tool which is used to simulate various conditions for checking the functionality of the application firmware.

Debugging process is a process through which a system performance can be diagnosed.

Debugging process can take one of the following apporach –

* Hardware debugging
* Firmware debugging

Emulator is a self-contained hardware device which emulates the target device. Emulator hardware contains necessary emulation logic and is connected to the target board through some interface.

Embedded product development lifecycle is a analysis – design – implementation approach for embedded system design.

This approach is necessary to understand scope and complexity of the work involved in any embedded product development.

EDLC or embedded product development lifecycle tries to ensure maximum benefit from the embedded product in following ways –

* Ensuring high quality product
* Risk minimizarion
* Increased Productivity

Different phases in embedded product development lifecycle are –

Requirement / Need of product, Conceptualization, Analysis, Design, Development and Testing, Deployment, support, Upgrade and Retirement.

Various phases of Embedded product lifecycle are interconnected in a way specified by a model.

Following are popular EDLC models –

* Waterfall or linear
* Incremental or iterative
* Prototype
* Spiral

Embedded system has evolved and have come up with many modification and changes due to advancements in technology.

* These evolutionary trends are in following categories –
* Processor Trends
* OS Trends
* Development Language Trends

Advantages in innovations in embedded systems have encouraged many manufacturers to form alliances and come up with better products.

However, there are no standards defined in embedded development, memory performance is not as good as processor or OS performance and very limited number of skilled resources are the bottlenecks in embedded development.

**Questions**

* Explain the steps in design and development of embedded system
* What are the constituents of embedded system development environment?
* What are the basic steps in using IDE tool keil µvision 5 to develop a embedded firmware project?
* What is meant by cross compilation? Name different types of files generated in cross compilation process.
* Explain the function of each file type generated in cross compilation.
* What are disassemblers and decompilers? What can be the possible use of these tools?
* What is simulator? Explain the advantages of simulator based debugging.
* Explain the difference between simulator and emulator.
* What are different debugging techniques? Discuss each in brief.
* Compare various firmware debugging approaches.
* List various phases in embedded product lifecycle development. Explain the function of each phase in detail.
* What is the use of EDLC models? List and explain different models used in embedded product development lifecycle.