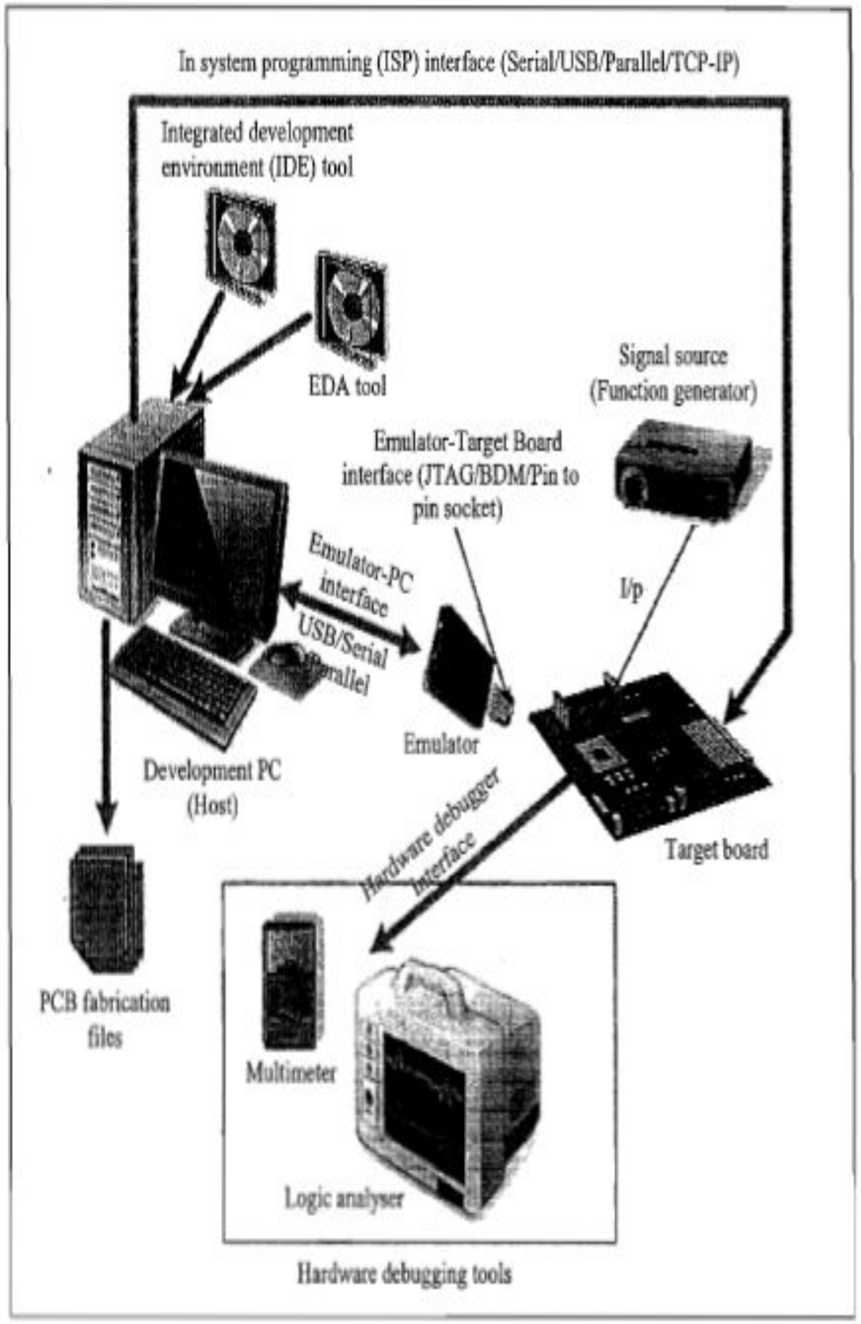
EMBEDDED SYSTEM DEVELOPMENT ENVIRONMENT


In system programming (ISP) interface (Serial/USB/Parallel/TCP-IP)
Integrated development environment (IDE) tool
EDA tool
Signal source (Function generator)
Emulator-Target Board interface (JTAG/BDM/Pin to pin socket)
Emulator-PC interface USB/Serial Parallel
I/p
Emulator
Development PC (Host)
Target board
Hardware debugger interface
PCB fabrication files
Multimeter
Logic analyser
Hardware debugging tools

The Embedded System Development Environment

Components of Embedded development environment

- Host Computer
  - Acts as the heart of development environment.
- IDE Tools
  - Tools for firmware development and debugging
- Electronic Design Automation (EDA) Tools
  - Embedded Hardware Design
- Emulator hardware
  - Debugging target board

Components of Embedded development environment

- Signal Sources (function generator)
  - Simulates inputs to target board
- Target Hardware Debugging tools
  - CRO, Multimeter ,Logic Analyser
  - For debugging hardware
- Target Hard ware

- The Integrated Development Environment (IDE) and Electronic Design Automation (EDA) tools are selected based on the target hardware development requirement
- They are supplied as Installable files in CDs/Online downloads by vendors.
- These tools need to be installed on the host PC used for development activities.
- These tools can be either freeware or licensed copy or evaluation versions.
- Licensed versions of the tools are fully featured and fully functional whereas trial versions fall into two categories, tools with limited features, and full featured copies with limited period of usage.

IDE

- IDE stands for Integrated Development Environment
- It is an integrated environment for developing and debugging the target processor specific embedded firmware.
- IDE is a software package which bundles a ‘Text Editor ‘,‘Cross-compiler ‘, ‘Linker’ and a ‘Debugger’
- An IDE is also known as integrated design environment or integrated debugging environment.
- IDE is a software application that provides facilities to computer programmers for software development.
- IDEs can either command line based or GUI based

- IDEs used in embedded firmware development are slightly different from the generic IDEs used for high level language based development for desktop applications.
- In Embedded Applications, the IDE is either supplied by the target processor/controller manufacturer or by third party vendors or as Open Source.
- **Examples:**
- Keil µVision5 is a licensed IDE tool from Keil Software (www.keil.com), for 8051 family microcontroller based embedded firmware development
- MPLAB is an IDE tool supplied by microchip for developing embedded firmware using their PIC family of microcontrollers. .
- CodeWarrior Development Studio is an IDE for ARM family of processors/MCUs and DSP chips from Freescale.

- In embedded firmware development applications each IDE is designed for a specific family of controllers/processors and it may not be possible to develop firmware for all family of controllers/processors using a single
- Eclipse is an open source IDE for embedded development.

IDE Components

1.Text Editor or Source code editor

2.A compiler and an interpreter

3.Build automation tools

4.Debugger

5.Simulators

6.Emulators and logic analyzer

E.g. Turbo C/C++,Microsoft visual c++ etc

Keil µVersion3/5 IDE for 8051

- To start with IDE,execute the Keil uversion3 from desktop.ie similar to microsoft visual studio IDE.
- The µVision IDE combines project management, run-time environment, build facilities, source code editing, and program debugging in a single powerful environment.
- µVision is easy-to-use and accelerates your embedded software development.
- µVision supports multiple screens and allows you to create individual window layouts anywhere on the visual surface.

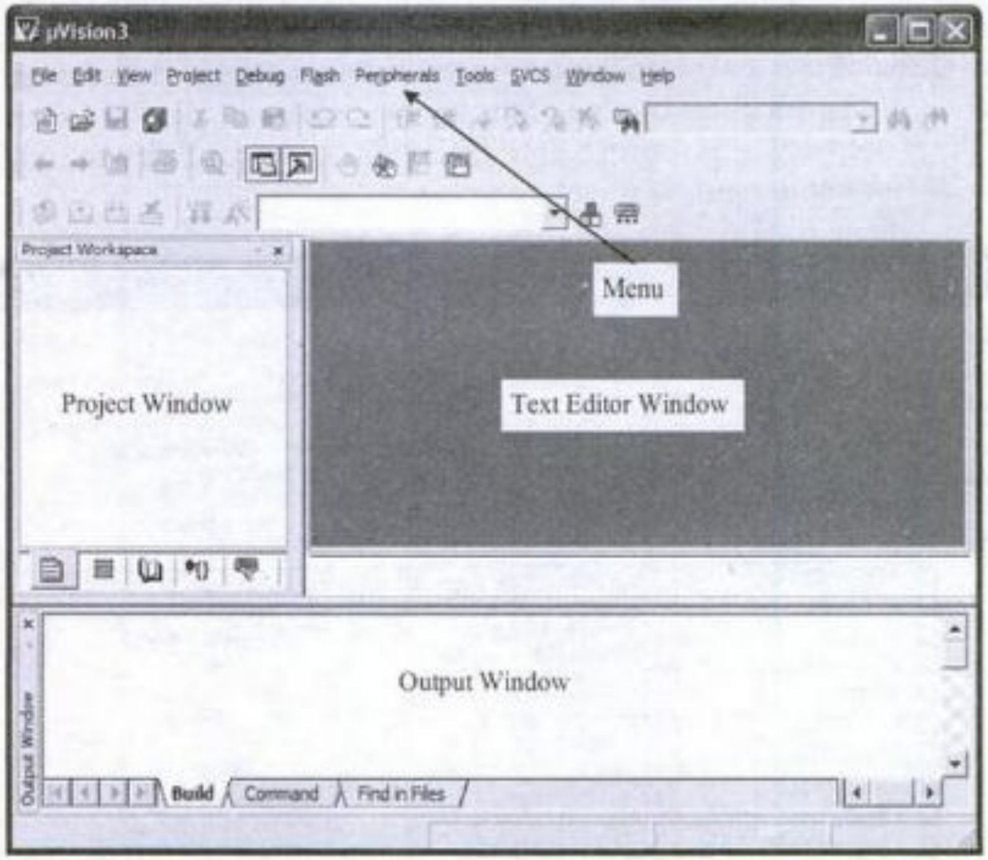
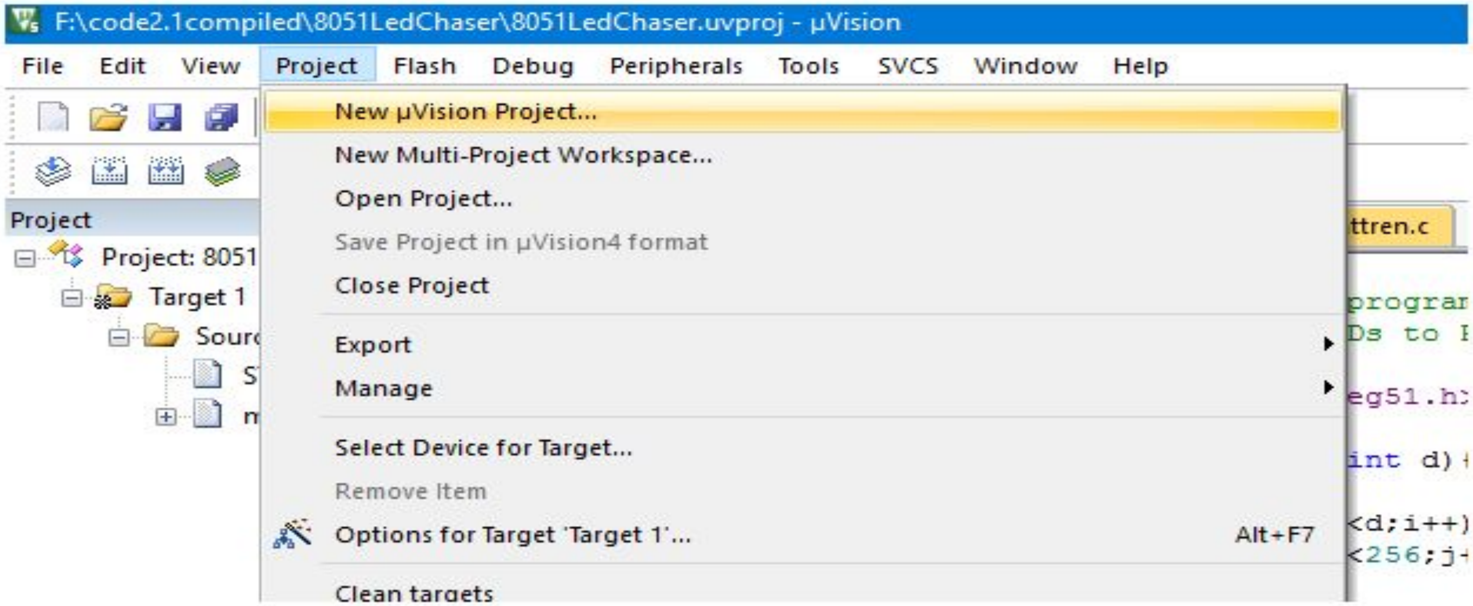


Fig. 13.2 Keil µVision3 Integrated Development Environment (IDE)

Step2:create a new project

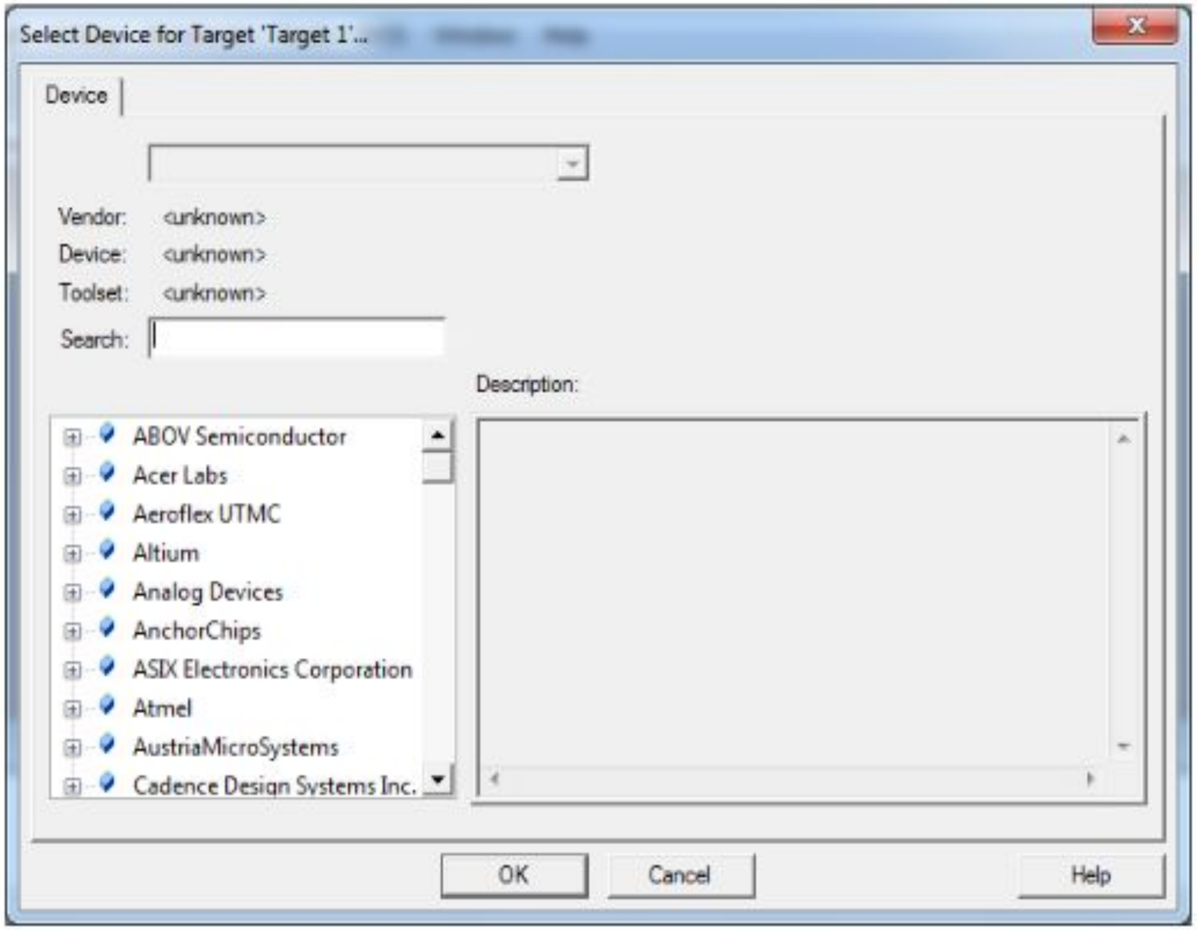
Creating Project in keil :

Select the project- > new uVision project


F:\code2.1compiled\8051LedChaser\8051LedChaser.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
New µVision Project...
New Multi-Project Workspace...
Open Project...
Save Project in µVision4 format
Close Project
Export
Manage
Select Device for Target...
Remove Item
Options for Target 'Target 1'... Alt+F7
Clean targets
Project
Project: 8051
Target 1

Once you select 'New uVision Project',it will prompt for a project name. Provide a suitable name for your project

Step3:select CPU vendor,device and toolkit for keil IDE

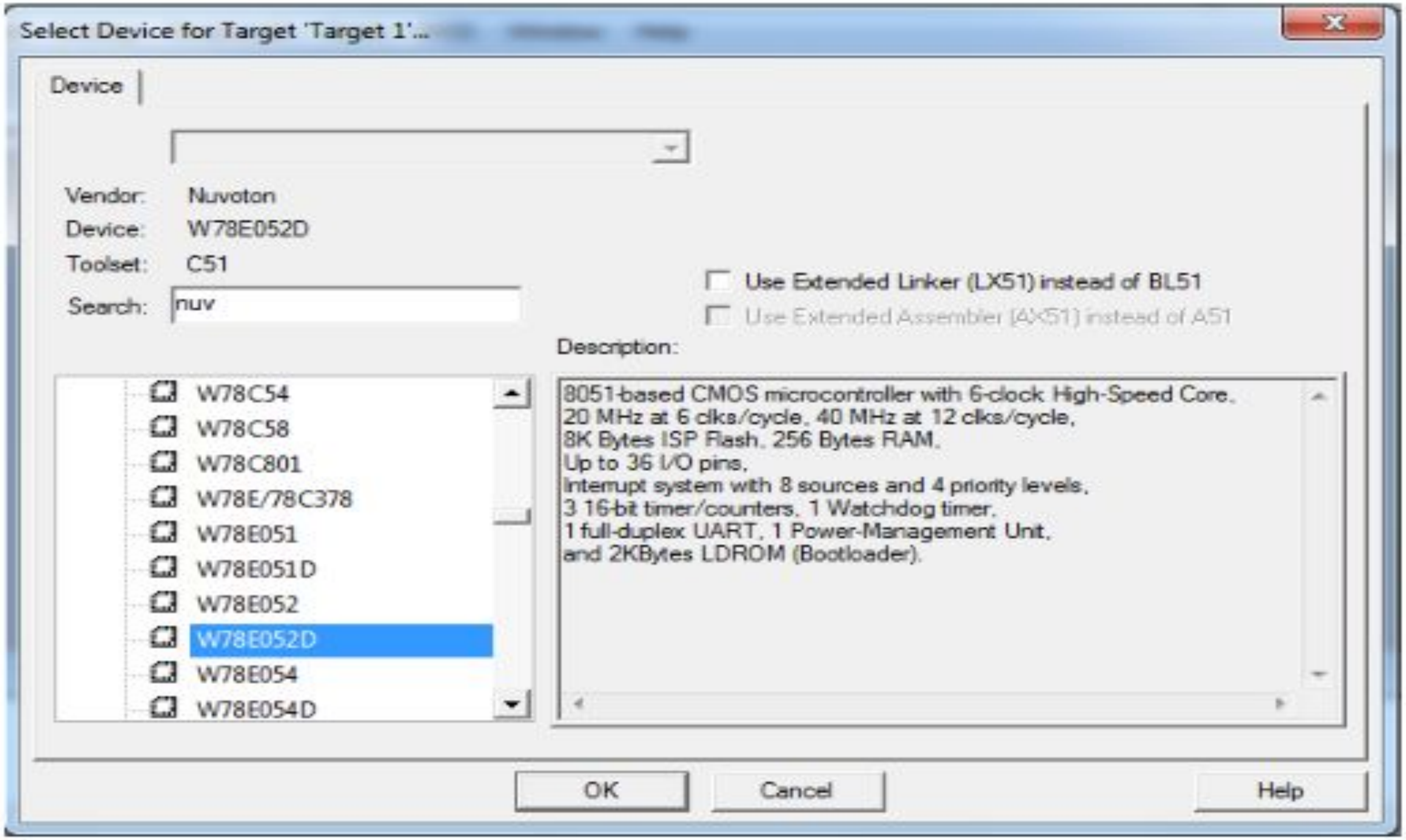


Vendor: <unknown>
Device: <unknown>
Toolset: <unknown>
Search:

Description:

- ABOV Semiconductor
- Acer Labs
- Aeroflex UTMC
- Altium
- Analog Devices
- AnchorChips
- ASIX Electronics Corporation
- Atmel
- AustriaMicroSystems
- Cadence Design Systems Inc.

Here i **am using W78E052D from Nuvoton** for my project.



Vendor: Nuvoton
Device: W78E052D
Toolset: C51
Search: nuv

- W78C54
- W78C58
- W78C801
- W78E/78C378
- W78E051
- W78E051D
- W78E052
- W78E052D
- W78E054
- W78E054D

[ ] Use Extended Linker (LX51) instead of BL51
[ ] Use Extended Assembler (AX51) instead of A51

Description:

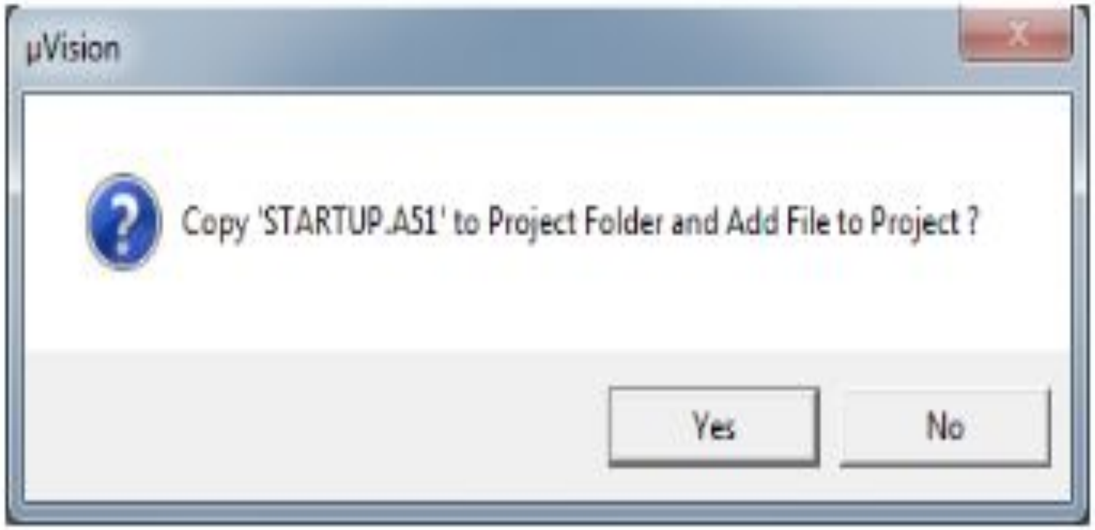
8051-based CMOS microcontroller with 6-clock High-Speed Core,
20 MHz at 6 clks/cycle, 40 MHz at 12 clks/cycle,
8K Bytes ISP Flash, 256 Bytes RAM,
Up to 36 I/O pins,
Interrupt system with 8 sources and 4 priority levels,
3 16-bit timer/counters, 1 Watchdog timer,
1 full-duplex UART, 1 Power-Management Unit,
and 2KBytes LDROM (Bootloader).

On selecting the particular microcontroller the Keil IDE also displays the features of the selected microcontroller on its left pane .You can Click *OK* to confirm your choice.

- Keil has support for a wide variety of 8051 derivatives on its IDE. The 8051 derivatives are organized according to their manufacturer's.
- E.g. Lets assume that you are developing code for ATMEL AT89S52 ,you can click on the ATMEL link on the bottom left pane and then browse for your chosen microcontroller here AT89S52.Alternatively you can also use the Search box on top to search for your particular part number.

Step 4:

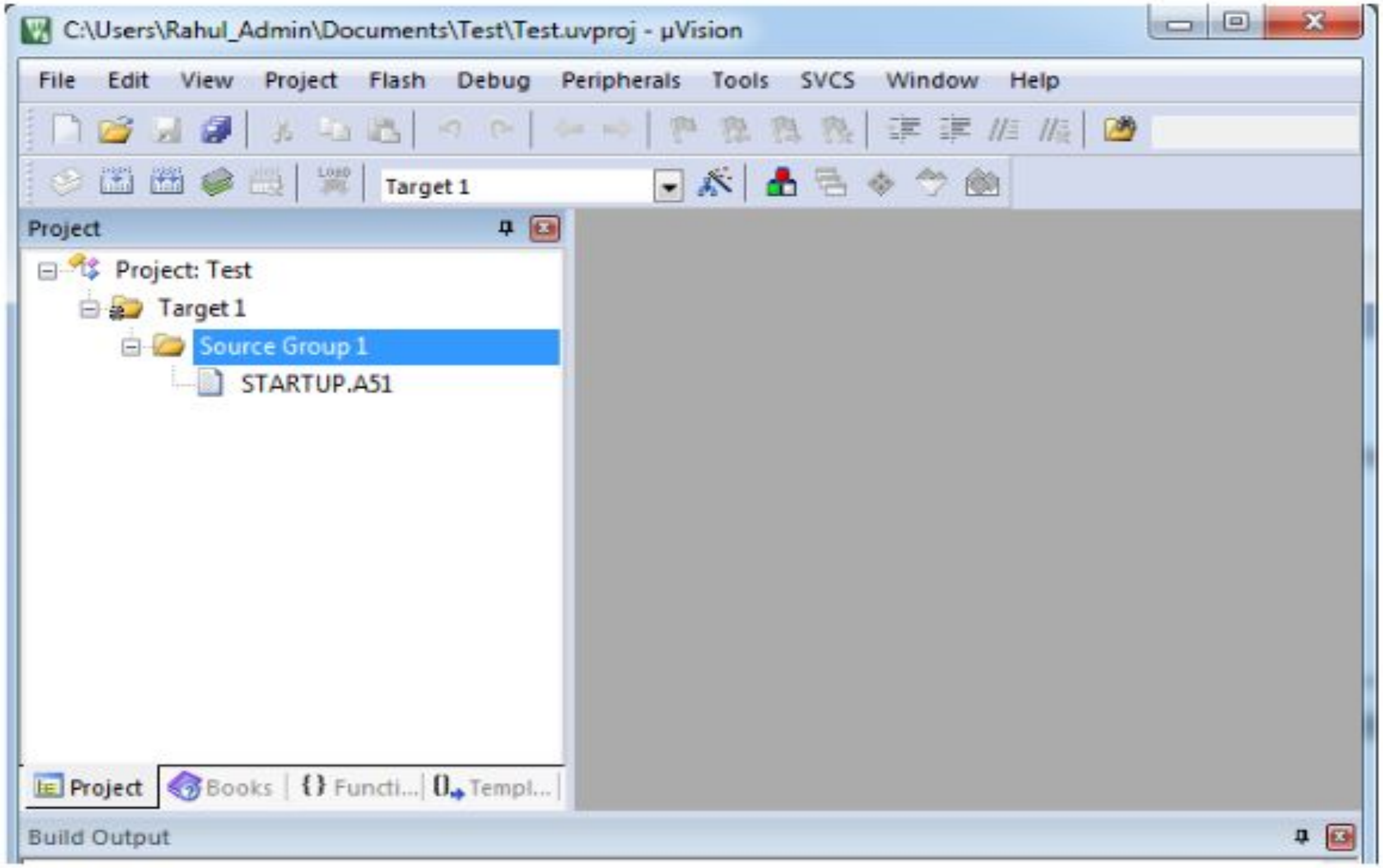
- After selecting your 8051 derivative,
- You will get another dialog as shown Above.Asking to copy STARTUP.A51
- Click ' Yes ‘


Copy 'STARTUP.A51' to Project Folder and Add File to Project ?

Click ' ***Yes*** '

Now your **Project pane** on the Kiel IDE would look something like this (below image)

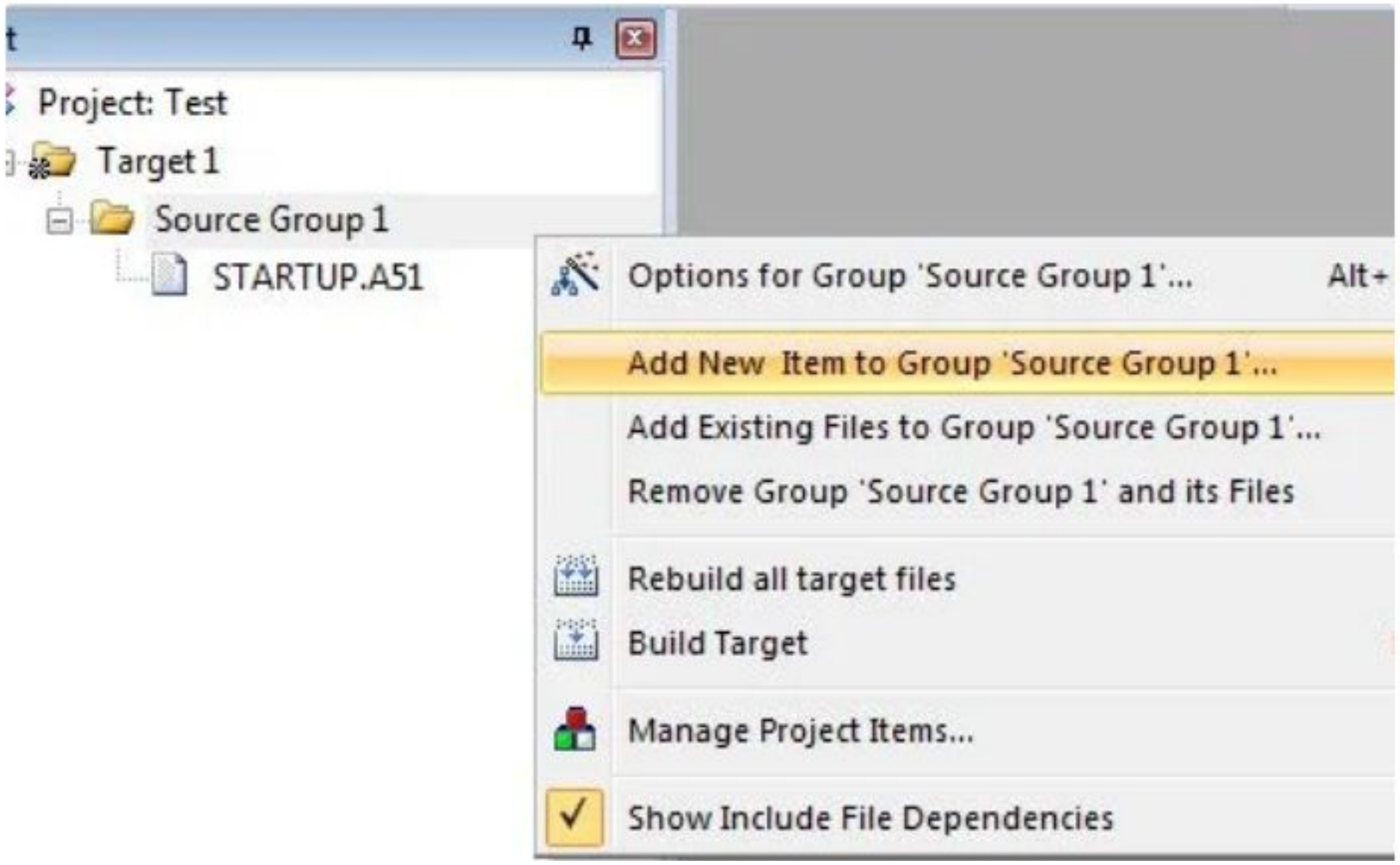
Now your **Project pane** on the Kiel IDE would look something like this (below image)


C:\Users\Rahul_Admin\Documents\Test\Test.uvproj - µVision
File Edit View Project Flash Debug Peripherals Tools SVCS Window Help
Target 1
Project
Project: Test
Target 1
Source Group 1
STARTUP.A51
Project Books Functi... Templ...
Build Output

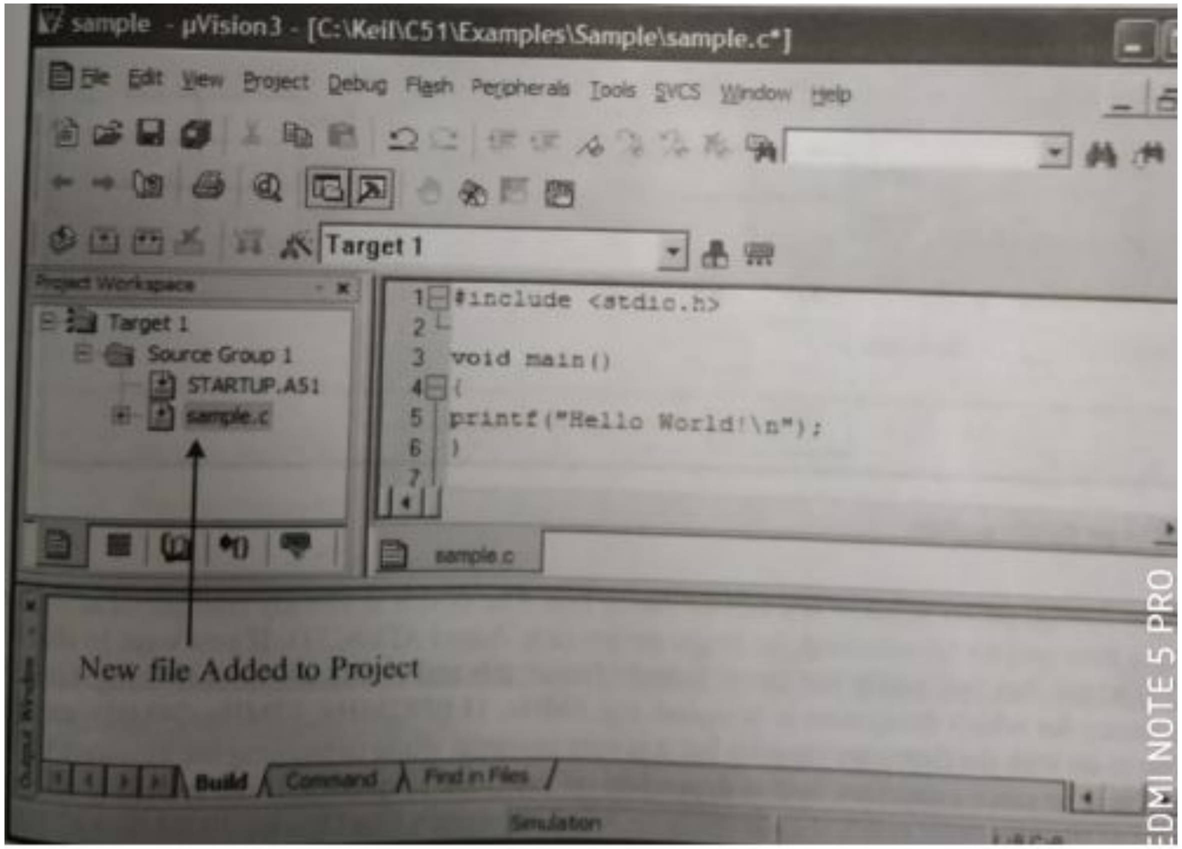
Now you can add C files to you Project.

- Here Target 1 Is automatically generated under Files section of project window
- This contain source group with the name ‘Source Group1’and ‘start up’ file is kept under this

Step6: adding C file to the project


t
Project: Test
Target 1
Source Group 1
STARTUP.A51
Options for Group 'Source Group 1'... Alt+
Add New Item to Group 'Source Group 1'...
Add Existing Files to Group 'Source Group 1'...
Remove Group 'Source Group 1' and its Files
Rebuild all target files
Build Target
Manage Project Items...
Show Include File Dependencies

Step 7: Here I am adding Sample .C to my project


sample - µVision3 - [C:\Keil\C51\Examples\Sample\sample.c*]
File Edit View Project Debug Flash Peripherals Tools SVCS Window Help
Target 1
Project Workspace
Target 1
Source Group 1
STARTUP.A51
sample.c
Build
Command
Find in Files
Simulation
REDMI NOTE 5 PRO

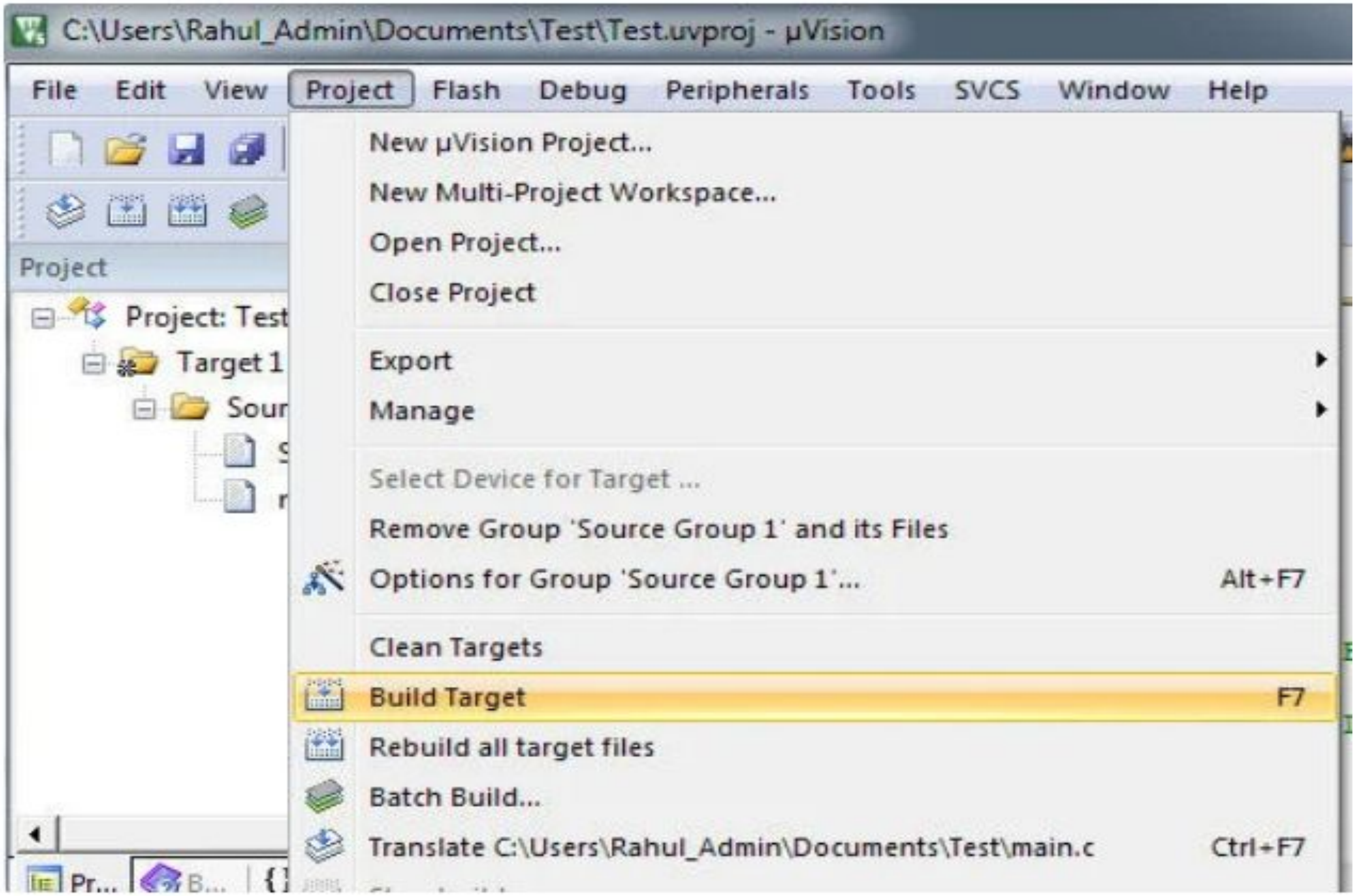
```
#include <stdio.h>

void main()
{
printf("Hello World!\n");
}
```

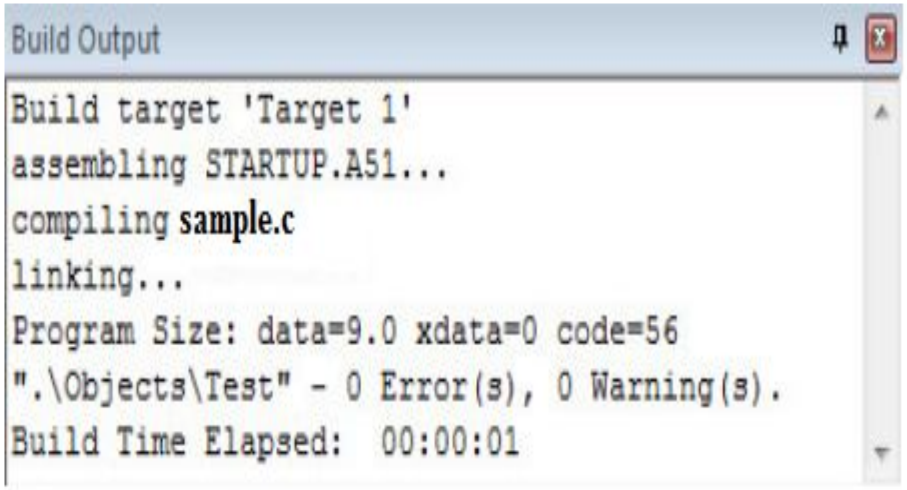
New file Added to Project

Step 8:

- After you have typed out the above c program to your sample.c file,You can compile the C file by pressing **F7 key** or by going to ' **Project -> Build Target** ' on the IDE menu bar.

Step 8: Building a C Project Using Keil UVision IDE


C:\Users\Rahul_Admin\Documents\Test\Test.uvproj - µVision
File
Edit
View
Project
Flash
Debug
Peripherals
Tools
SVCS
Window
Help
Project
Project: Test
Target 1
New µVision Project...
New Multi-Project Workspace...
Open Project...
Close Project
Export
Manage
Select Device for Target ...
Remove Group 'Source Group 1' and its Files
Options for Group 'Source Group 1'... Alt+F7
Clean Targets
Build Target F7
Rebuild all target files
Batch Build...
Translate C:\Users\Rahul_Admin\Documents\Test\main.c Ctrl+F7

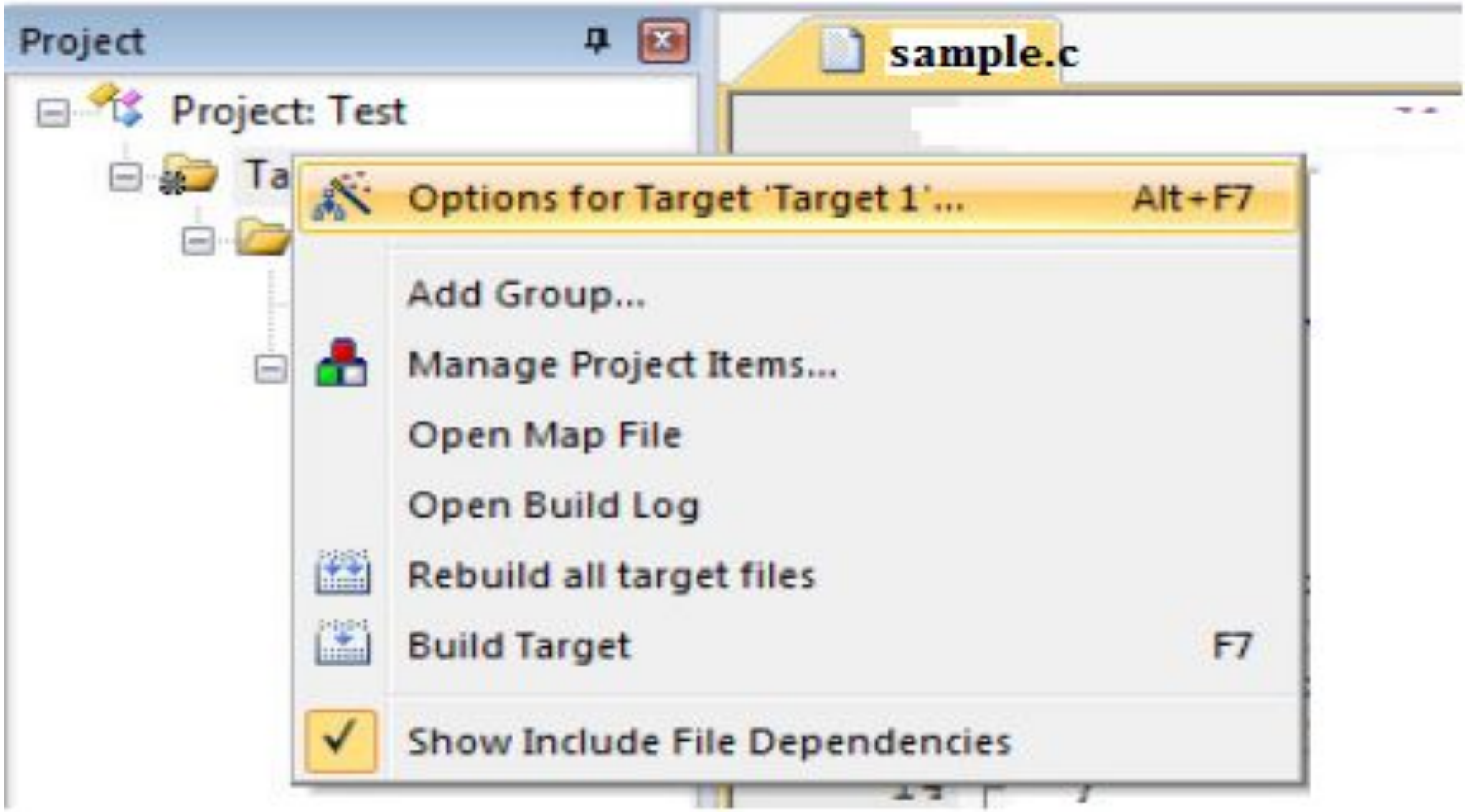


```
Build target 'Target 1'
assembling STARTUP.A51...
compiling sample.c
linking...
Program Size: data=9.0 xdata=0 code=56
".\Objects\Test" - 0 Error(s), 0 Warning(s).
Build Time Elapsed:  00:00:01
```

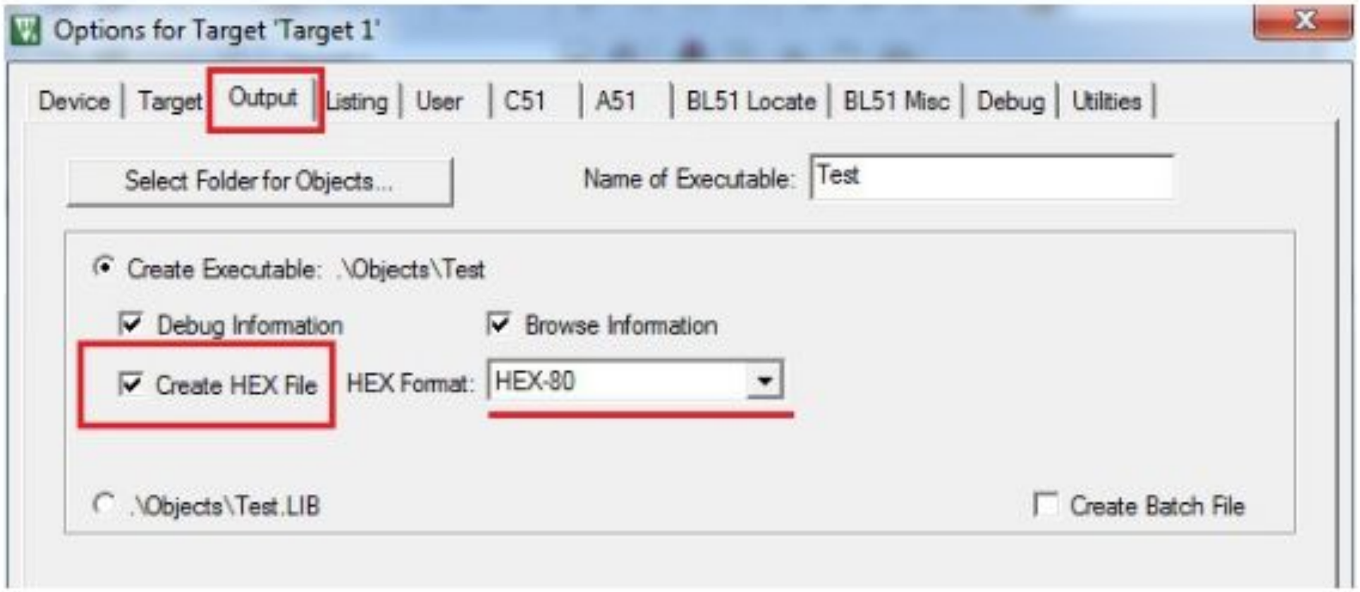
- If there are no errors the code will compile and you can view the output on the ***Build Output*** pane

step 9:Generating 8051 HEX File using Kiel IDE

- In the above example we have only compiled our sample .c file.Inorder to download the code into the 8051 microcontroller we have to generate the corresponding hexcode .
- In Keil uVision IDE you can generate hexfile for your 8051 derivative by,
- Right Clicking on the ***' Target 1 '*** Folder and Selecting ***Options for Target 'Target1'....***



Then on the ***Options for Target ' Target 1'*** Dialog , Select the ***Output*** tab and check the **Create Hex File** option and Press ***OK***.


Options for Target 'Target 1'

Device | Target | Output | Listing | User | C51 | A51 | BL51 Locate | BL51 Misc | Debug | Utilities

Select Folder for Objects...

Name of Executable: Test

(●) Create Executable: .\Objects\Test

[✓] Debug Information

[✓] Browse Information

[✓] Create HEX File

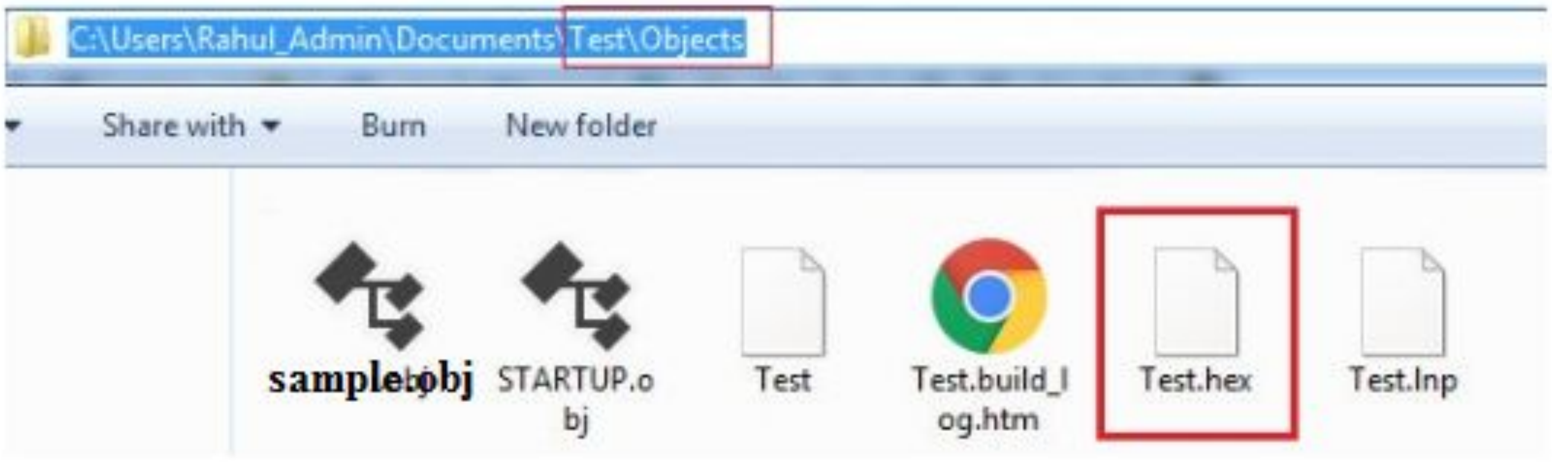
HEX Format: HEX-80

( ) .\Objects\Test.LIB

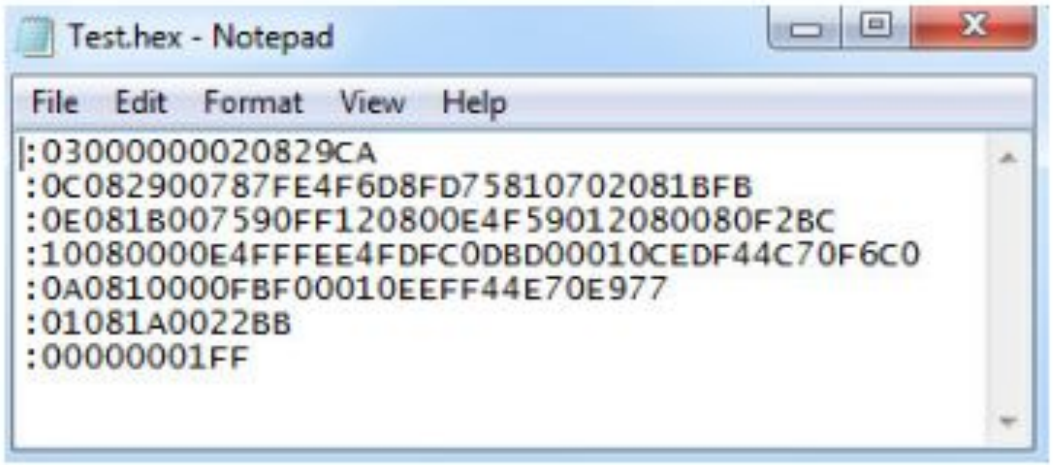
[ ] Create Batch File

Now rebuild your project by pressing F7.

Kiel IDE would generate a hex file with same name (here Test.hex) as your project in the ***Objects folder*** .



You can also open the Test.hex file with notepad to view the contents.



```
:0300000020829CA
:0C082900787FE4F6D8FD75810702081BFB
:0E081B007590FF120800E4F59012080080F2BC
:10080000E4FFFEE4FDFC0DBD00010CEDF44C70F6C0
:0A0810000FBF00010EEFF44E70E977
:01081A0022BB
:00000001FF
```

CROSS COMPILERS

TYPES OF FILES GENERATED ON CROSS-COMPILATION

- Cross-compilation is the process of converting a source code written in high level language to a target processor/ controller understandable machine code
- The conversion of the code is done by a software referred as the 'Cross-compiler'.
- Cross-compilation is the process of cross platform software/firmware development.
- The application converting Assembly instruction to target processor / controller specific machine code is known as cross-assembler.

- Cross-compilation/cross-assembling is carried out in different steps and the process generates various types of intermediate files.
- The various files generated during the cross compilation/cross-assembling process

Types of Files Generated on Cross compilation

1.List File

2.Pre-processor Output file

3.Hex File (.hex)

4.Map File (File extension linker dependent)

5.Object File (.obj)

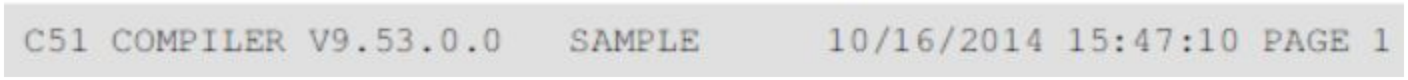
1.List Files(.LST files)

- Generated at the time of cross compilation
- Contain information about cross compilation process like
  - Cross compiler details
  - Formatted source text
  - Assembly code generated from the source file
  - Symbol table
  - Errors and warning detected by the cross compiler system
- The type of information contained in the list file is cross-compiler specific.

consider the cross-compilation process of the file sample.c given under Keil µVision5 IDE

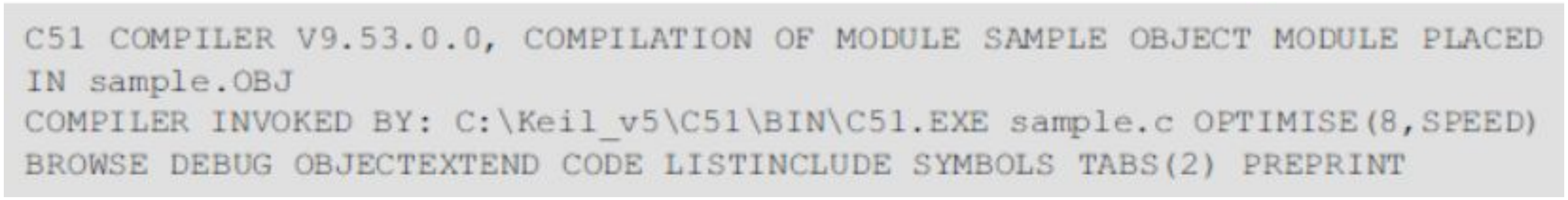
- The ‘list file’ generated contains the following sections
  - Page Header
  - Command Line
  - Source Code
  - Assembly Listing
  - Symbol Listing
  - Module Information
  - Warnings and Errors

- **Page header** : A header on each page of the listing file which indicates the compiler version number, source file name, date, time, and page number.



```
C51 COMPILER V9.53.0.0    SAMPLE        10/16/2014 15:47:10 PAGE 1
```

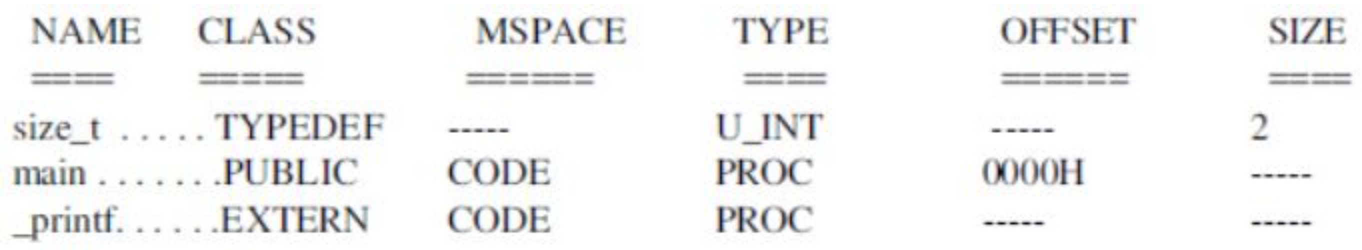
- **Command line** :Represents the entire command line that was used for invoking the compiler.



```
C51 COMPILER V9.53.0.0, COMPILATION OF MODULE SAMPLE OBJECT MODULE PLACED
IN sample.OBJ
COMPILER INVOKED BY: C:\Keil_v5\C51\BIN\C51.EXE sample.c OPTIMISE(8,SPEED)
BROWSE DEBUG OBJECTEXTEND CODE LISTINCLUDE SYMBOLS TABS(2) PREPRINT
```

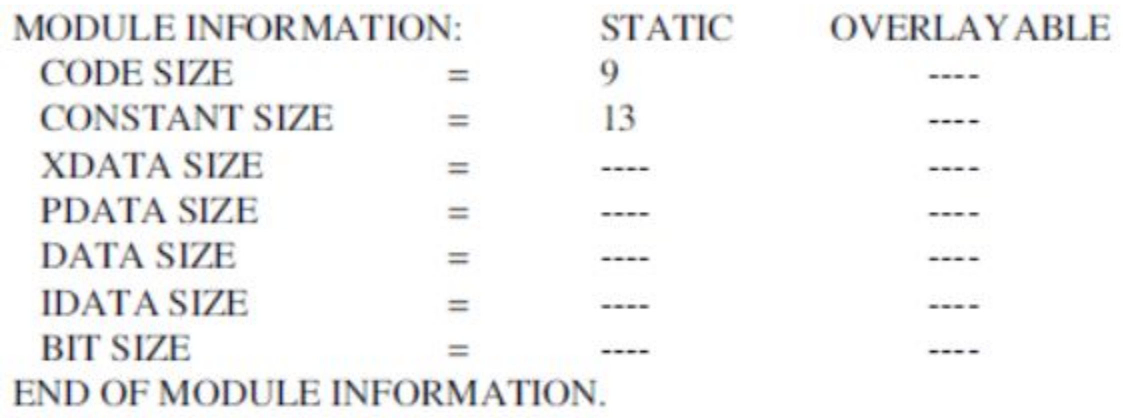
- **Source Code** :
  - The source code listing outputs the line number as well as the source code on that line.
  - Special cross compiler directives can be used to include or exclude the conditional codes (code in #if blocks) in the source code listings.
  - The list file will include the comments in the source file

- **Assembly Listing**
  - Contains the assembly code generated by the cross compiler for the ‘C’ source code.
  - Can be excluded from the list file by using special compiler directives.
- **Symbol Listing**
  - contains symbolic information about the various symbols present in the cross compiled source file.
  - Symbol listing contains the sections :
    - NAME - symbol name
    - CLASS - symbol classification ie, Special Function Register (SFR), structure, typedef, static, public, auto, extern, etc.,
    - MSPACE - memory space ie, code memory or data memory
    - TYPE - data type ie, int, char, Procedure call, etc.
    - OFFSET - offset from code memory start address
    - SIZE size in bytes ().
  - Symbol listing in list file output can be turned on or off by cross-compiler directives.



```
NAME      CLASS         MSPACE      TYPE        OFFSET      SIZE
====      =====         ======      ====        ======      ====
size_t . . . . . TYPEDEF   -----       U_INT       -----       2
main . . . . . . . .PUBLIC   CODE        PROC        0000H       -----
_printf. . . . . .EXTERN   CODE        PROC        -----       -----
```

- Module Information:
  - The module information provides the size of initialised and un-initialised memory areas defined by the source file



```
MODULE INFORMATION:          STATIC      OVERLAYABLE
   CODE SIZE            =    9               ----
   CONSTANT SIZE        =    13              ----
   XDATA SIZE           =    ----            ----
   PDATA SIZE           =    ----            ----
   DATA SIZE            =    ----            ----
   IDATA SIZE           =    ----            ----
   BIT SIZE             =    ----            ----
END OF MODULE INFORMATION.
```

- Warnings and Errors :
  - This section of list file records the errors encountered or any statement that may create issues in application (warnings), during cross compilation.
  - The warning levels can be configured before cross compilation.
  - certain warnings can be ignored certain requires prompt attention.
- List file is a very useful tool for application debugging in case of any cross compilation issues.

2.Preprocessor output file

- Generated during cross compilation
- Contain preprocessor output for the preprocessor instructions used in the source file
- This file is used for verifying the operation of macros and conditional preprocessor directives
- This file is a valid C file
- File extension is cross compiler dependent

3. Object files

- Cross compiling each source module converts the various Embedded instructions and other directives present in the module to an object(.OBJ) file.
- The format of the .OBJ fi le is cross compiler dependent.
- OMF51 or OMF2 are the two objects file formats supported by C51 cross compiler.
- The object file is specially formatted file with data records for symbolic information, object code, debugging information etc.

- The list of some of the details stored in an object file is given below.
  1. Reserved memory for global variables.
  2. Public symbol (variable and function) names.
  3. External symbol (variable and function) references.
  4. Library files with which to link.
  5. Debugging information to help synchronise source lines with object code.
- code is not allocated fixed memory location in code memory.
- It is the responsibility of the linker/locater to assign an absolute memory location to the object code.
- During cross-compilation process, the cross compiler sets the address of references to external variables and functions as 0.
- The external references are resolved by the linker during the linking process.

4.Map Files(.MAP)

- Object file created contains re-locatable codes where their location in memory is not fixed
- It is the responsibility of **linker** to link these object modules
- The **locator** is responsible for locating the absolute address to each module in the code memory

- Linking and locating of re-locatable object files will also **generate a list file called 'linker list file' or 'map file'.**
- Map file contains information about the link/locate process and is composed of a number of sections.
- The different sections listed in a map file are cross compiler dependent
- It includes:

- **Page Header : include** the linker version number, date, time, and page number.
- **Command Line :** Represents the entire command line that was used for invoking the linker
- **CPU Details** :Details about the target CPU and memory model (internal data memory, external data memory, paged data memory, etc.) come under this category
- **Input Modules :**This section includes the names of all object modules, and library files and modules that are included in the linking process.
- **Memory Map**: Memory map lists the starting address, length, relocation type and name of each segment in the program
- **Symbol Table :** It contains the value, type and name for all symbols from the different input modules

- **Inter Module Cross Reference:** includes the 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:** contain the size of various memory areas as well as constant and code space for the entire application
- **Warnings and Errors:** Errors and warnings generated while linking process

5. Hex file

- Hex file is the **binary executable file** created from the source code.
- The absolute object file created by the linker/locater is converted into processor understandable binary code.
- The utility used for converting an object file to a hex file is known as **Object to Hex file converter**.
- The format of Hex file varies across the family of processors/controllers.
- Intel HEX and Motorola HEX are the two commonly used hex file formats in embedded applications.
- Intel HEX file is an ASCII text file in which the HEX data is represented in ASCII format in lines.

- The lines in an Intel HEX file are corresponding to a HEX Record.
- Each record is made up of hexadecimal numbers that represent machine-language code and/or constant data.
- Individual records are terminated with a carriage return and a linefeed.
- Intel HEX file is used for transferring the program and data to a ROM or EPROM which is used as code memory storage

Disassembler / Decompiler

Disassembler / Decompiler

- **Disassembler**
  - Disassembler is a utility program which converts machine codes into target processor specific Assembly codes / instructions.
  - The process of converting machine codes into Assembly code is known as 'Disassembling'.
  - In operation, disassembling is complementary to assembling / crossassembling
- **Decompiler**
  - Is a utility program that convert machine language instruction to high level language instruction
  - Performs reverse operation of compiler or cross compiler

Disassembler / Decompiler cont.

- Both are **reverse engineering** tools
- Reverse engineering is the process of revealing the technology behind the working of a product
- Used to find out the secret behind popular proprietary products
- Helps the reverse engineering process by translating embedded firmware to assembly /high level instruction
- Powerful tools for analyzing the presence of malicious codes (virus information) in an executable image
- These are available as either freeware tools readily available for free download from internet or as commercial tools

Simulators

SIMULATORS

- Simulators and emulators are two important tools used in embedded system development
- Simulator is a software tool for simulating various conditions for

  checking the functionality of the application firmware
- Simulator help in debugging the firmware.
- IDE provides simulator support

FEATURES OF SIMULATOR BASED DEBUGGING

- Purely software based
- Doesn’t require a real target system
- Very primitive (Lack of featured I/O support.)
- Lack of real time behavior

Advantage of simulator based debugging

- **No need for original target board**
  - Purely software oriented , IDE simulates the target board
  - Since real hardware is not needed, firmware development can start immediately after the device interface and memory maps are finalized, this saves development time
- **Simulated I/O peripherals**
  - Simulator provides the option to simulate various I/O peripherals
  - Using simulator's I/O support you can edit the values for I/O registers and can be used as the input/output value in the firmware execution
- It eliminates the need for connecting IO devices for debugging the firmware

- **Simulates abnormal conditions**
  - Can input any desired value for any parameter during debugging the firmware and can observe the control flow of firmware
  - It really helps the developer in simulating abnormal operational environment for firmware
  - It helps the firmware developer to study the behaviour of the firmware under abnormal input conditions

LIMITATIONS:

- we cannot fully rely upon the simulator-based firmware debugging
- **Deviation from real behavior**
  - Simulation-based firmware debugging is always carried out in a development environment where the developer may not be able to debug the firmware under all possible combinations of input.
  - Under certain operating conditions we may get some particular result and it need not be the same when the firmware runs in a production environment
- **Lack of real timeliness**
  - The debugging is developer driven and it is no way capable of creating a real time behaviour.
  - Moreover in a real application the I/O condition may be varying or unpredictable.

Emulators and Debuggers

What is debugging?

- Debugging in embedded application is the process of diagnosing the firmware execution, monitoring the target processor's registers and memory while the firmware is running and checking the signals from various buses of the embedded hardware.
- Debugging is classified into two namely Hardware debugging and firmware debugging.
- Hardware debugging deals with the monitoring of various bus signals and checking the status lines of the target hardware.
- various tools used for hardware debugging
- Firmware debugging deals with examining the firmware execution, execution flow, changes to various CPU registers and status registers on execution of the firmware to ensure that the firmware is running as per the design.

Why is debugging required?

- Firmware debugging is performed to figure out the bug or the error in the firmware which createsthe unexpected behaviour
- A debugger or debugging tool is a computer program that is used to test and debug other programs.
- A debugger allows a programmer to stop a program at any point and examine and change the values of the variables.
- various types of firmware debugging techniques are available today
- Following techniques are improvements over firmware debugging starting from the most primitive type of debugging to the most sophisticated
  1. Incremental EEPROM Burning Technique
  2. Inline Breakpoint Based Firmware Debugging
  3. Monitor Program Based Firmware Debugging
  4. In Circuit Emulator (ICE) Based Firmware Debugging
  5. On Chip Firmware Debugging (OCD)

Incremental EEPROM Burning Technique

- Most primitive type of firmware debugging technique
- Here code is separated into different functional code units.
- Instead of burning the entire code into the EEPROM chip at once, the code is burned in incremental order
- Code corresponding to all functionalities are separately coded, cross-compiled and burned into the chip one by one.
- The code will incorporate some indication support like lighting up an "LED" it can be used for debugging purpose or activate a "BUZZER (In a system with BUZZER support)" if the code is functioning in the expected way.
- If the first functionality is found working perfectly on the target board with the corresponding code burned into the EEPROM

- Then go for burning the code corresponding to the next functionality and check whether it is working.
- Repeat this process till all functionalities are covered.
- If the code for any functionality is found not giving the expected result, fix it by modifying the code and then only go for adding the next functionality for burning into the EEPROM.
- After you found all functionalities working properly, combine the entire source for all functionalities together, re-compile and burn the code for the total system functioning.
- It is a onetime process and once you test the firmware in an incremental model you can go for mass production
- In incremental firmware burning technique we are not doing any debugging but observing the status of firmware execution as a debug method.

- The very common mistake committed by firmware developers in developing non-operating system-based embedded application is burning the entire code altogether and fed up with debugging the code.
- Even though this approach is time consuming, you will never lose in the process and will never mess up with debugging the code.
- You will be able to figure out at least 'on which point of firmware execution the issue is arising
- widely adopted in small, simple system developments and in product development where time is not a big constraint (e.g. R&D projects).
- useful in product development environments where no other debug tools are available.

Inline Breakpoint Based Firmware Debugging

- Another primitive method of firmware debugging.
- Within the firmware where you want to ensure that firmware execution is reaching up to a specified point, insert an inline debug code immediately after the point.
- The debug code is a printf() function which prints a string given as per the firmware.
- You can insert debug codes (printf()) commands at each point where you want to ensure the firmware execution is covering that point.
- Cross-compile the source code with the debug codes embedded within it.

- Burn the corresponding hex file into the EEPROM. You can view the printf() generated data on the a ‘Terminal ’
- If the firmware is error free and the execution occurs properly, you will get all the debug messages on the Terminal . Based on this debug info you can check the firmware for errors

Monitor Program Based Firmware Debugging

- In this approach, develop a monitor program which act as a supervisor
- The monitor program controls the downloading of user code into the code memory, inspects and modifies register/memory locations; allows single stepping of source code, etc
- The monitor program implements the debug functions as per a pre-defined command set from the debug application interface
- The monitor program always listens to the serial port of the target device and according to the command received from the serial interface
- It performs command specific actions like firmware downloading, memory inspection/modification, firmware single stepping and sends the debug information (various register and memory contents) back to the main debug program running on the development PC

- The first step in any monitor program development is determining a set of commands for performing various operations
- Commands may be received through any of the external interface of the target processor
- The monitor program should query this interface to get commands or should handle the command reception if the data reception is implemented through interrupts
- On receiving a command, examine it and perform the action corresponding to it.
- The entire code stuff handling the command reception and corresponding action implementation is known as the **“monitor program”.**

- The most common type of interface used between target board and debug application is RS-232/USB Serial interface.
- After the successful completion of the ‘monitor program’ development, it is compiled and burned into the FLASH memory or ROM of the target board.
- The code memory containing the monitor program is known as the‘ **Monitor ROM’.**
- The **monitor program** contains the following set of minimal features.

1. Command set interface to establish communication with the debugging application
2. Firmware download option to code memory
3. Examine and modify processor registers and working memory (RAM)
4. Single step program execution
5. Set breakpoints in firmware execution
6. Send debug information to debug application running on host machine

In Circuit Emulator (ICE) Based Firmware Debugging

EMULATOR

- Emulator is hardware device which emulates the functionalities of the target device and allows real time debugging of the embedded firmware in a hardware environment.
- It is a piece of hardware that exactly behaves like the real microcontroller chip with all its integrated functionality.
- It is the most powerful debugging of all.
- A microcontroller's functions are emulated in real-time and non-intrusively.
- An emulator is a piece of hardware that looks like a processor, has memory like a processor, and executes instructions like a processor but it is not a processor.
- The advantage is that we can probe points of the circuit that are not accessible inside a chip.
- It is a combination of hardware and software.

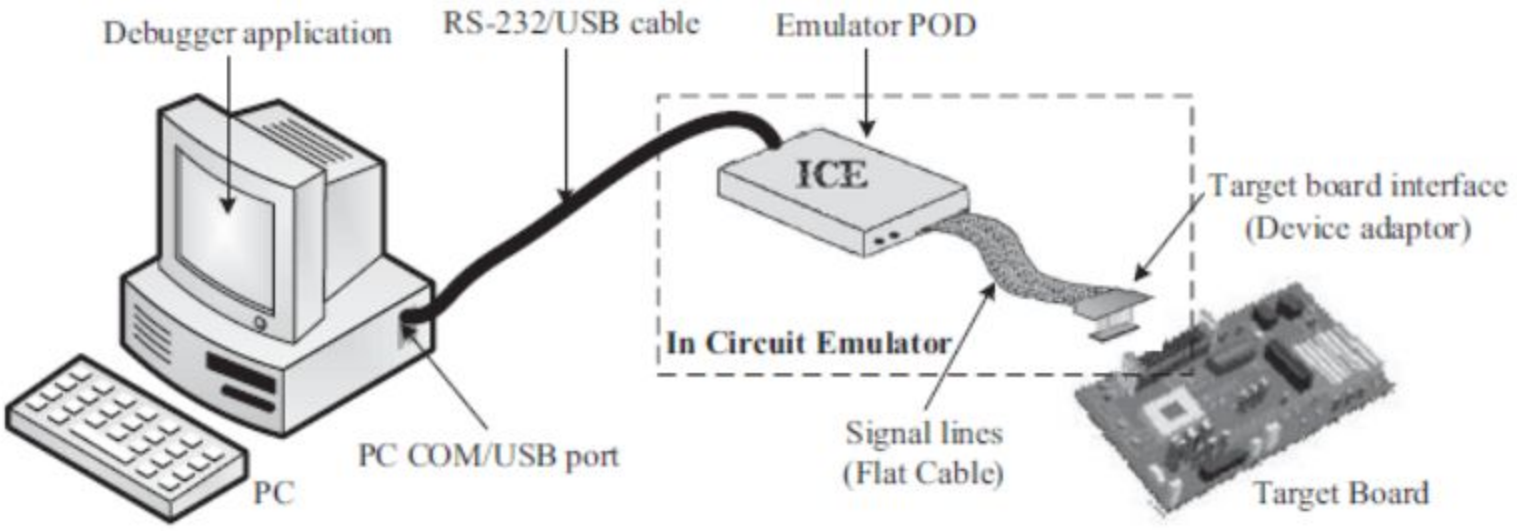


Fig. 13.40 In Circuit Emulator (ICE) Based Target Debugging

- The Emulator POD forms the heart of any emulator system and it contains the following functional units.
  - Emulation Device
  - Emulation Memory
  - Emulator Control Logic
  - Device Adaptors

Emulation Device

- Emulation device is a replica of the target CPU which receives various signals from the target board through a device adaptor
- It performs the execution of firmware under the control of debug commands from the debug application.
- It can be either a standard chip same as the target processor (e.g. AT89C51) or a PLD configured to function as the target CPU.
- If a standard chip is used as the emulation device, the emulation will provide real-time execution behaviour.
- At the same time the emulator becomes dedicated to that particular device and cannot be re-used for the derivatives of the same chip.
- PLD-based emulators can easily be re-configured to use with derivatives of the target CPU under consideration.
- By simply loading the configuration file of the derivative processor/controller, the PLD gets re-configured and it functions as the derivative device.
- A major drawback of PLD-based emulator is the accuracy of replication of target CPU functionalities.
- PLD-based emulator logic is easy to implement for simple target CPUs but for complex target CPUs it is quite difficult.

Emulation Memory

- It is the Random Access Memory (RAM) incorporated in the Emulator device.
- It acts as a replacement to the target board's EEPROM where the code is supposed to be downloaded after each firmware modification.
- Hence the original EEPROM memory is emulated by the RAM of emulator. This is known as 'ROM Emulation'.
- ROM emulation eliminates the hassles of ROM burning and it offers the benefit of infinite number of reprogramming (Most of the EEPROM chips available in the market supports only a few 1000 re-program cycles).
- Emulation memory also acts as a trace buffer in debugging.
- Trace buffer is a memory pool holding the instructions executed/registers modified/related data by the processor while debugging.
- The trace buffer size is emulator dependent and the trace buffer holds the recent trace information when the buffer overflows.

- The common features of trace buffer memory and trace buffer data viewing arelisted below:
  - Trace buffer records each bus cycle in frames
  - Trace data can be viewed in the debugger application as Assembly/Source code
  - Trace buffering can be done on the basis of a Trace trigger (Event)
  - Trace buffer can also record signals from target board other than CPU signals (Emulator dependent)
  - Trace data is a very useful information in firmware debugging

Emulator Control Logic

- Emulator control logic is the logic circuits used for implementing complex hardware breakpoints, trace buffer trigger detection, trace buffer control, etc.
- Emulator control logic circuits are also used for implementing logic analyser functions in advanced emulator devices.

Device Adaptors

- Device adaptors act as an interface between the target board and emulator POD.
- Device adaptors are normally pin-to-pin compatible sockets which can be inserted/plugged into the target board for routing the various signals from the pins assigned for the target processor.
- The device adaptor is usually connected to the emulator POD using ribbon cables.
- The adaptor type varies depending on the target processor's chip package.
- DIP, PLCC, etc. are some commonly used adaptors.

On Chip Firmware Debugging (OCD)

- Today almost all processors/controllers incorporate built in debug modules called On Chip Debug (OCD) support
- OCD- supports fast and efficient firmware debugging
- chip vendor dependent and most of them are proprietary technologies like Background Debug Mode (BDM), etc.
- Some vendors add 'on chip software debug support' through JTAG (Joint Test Action Group) port.
- Processors/controllers with OCD support incorporate a dedicated debug module to the existing architecture.
- Usually the on-chip debugger provides the means to set simple breakpoints, query the internal state of the chip and single step through code.

- OCD module implements dedicated registers for controlling debugging.
- An On Chip Debugger can be enabled by setting the OCD enable bit (The bit name and register holding the bit varies across vendors).
- Debug related registers are used for debugger control (Enable/disable single stepping, Freeze execution, etc.) and breakpoint address setting.
- BDM and JTAG are the two commonly used interfaces to communicate between the Debug application running on Development PC and OCD module of target CPU

TARGET HARDWARE DEBUGGING

- Hardware debugging involves the
  - monitoring of various signals of the target board (address/data lines, port pins, etc.),
  - checking the inter-connection among various components, circuit continuity checking, etc.
- The various hardware debugging tools used in Embedded Product Development are

Magnifying Glass (Lens)

- magnifying glass is the primary hardware debugging tool for an embedded hardware debugging professional.
- A magnifying glass is a powerful visual inspection tool.
- With a magnifying glass (lens), the surface of the target board can be examined thoroughly for dry soldering of components, missing components, improper placement of components, improper soldering,track (PCB connection) damage, short of tracks, etc

Multimeter

- A multimeter is used for measuring various electrical quantities like voltage (Both AC and DC), current (DC as well as AC), resistance, capacitance, continuity checking, transistor checking, cathode and anode identification of diode, etc.
- Any multimeter will work over a specific range for each measurement.
- It is the primary debugging tool for physical contact based hardware debugging
- In embedded hardware debugging it is mainly used for checking the circuit continuity between different points on the board, measuring the supply voltage, checking the signal value, polarity, etc.

Digital CRO

- CRO is used for waveform capturing and analysis, measurement of signal strength, etc.
- By connecting the point under observation on the target board to the Channels of the Oscilloscope, the waveforms can be captured and analysed for expected behaviour.
- CRO is a very good tool in analysing interference noise in the power supply line and other signal lines.

Logic Analyser

- Logic analyser is used for capturing digital data (logic 1 and 0) from a digital circuitry whereas CRO is employed in capturing all kinds of waves including logic signals.
- Another major limitation of CRO is that the total number of logic signals/waveforms that can be captured with a CRO is limited to the number of channels.
- A logic analyser contains special connectors and clips which can be attached to the target board for capturing digital data.
- In target board debugging applications, a logic analyser captures the states of various port pins, address bus and data bus of the target processor/ controller, etc.
- Logic analysers give an exact reflection of what happens when a particular line of firmware is running.
- This is achieved by capturing the address line logic and data line logic of target hardware

Function Generator

- Function generator is not a debugging tool.
- It is an input signal simulator tool.
- A function generator is capable of producing various periodic waveforms like sine wave, square wave, saw-tooth wave, etc. with different frequencies and amplitude.
- Sometimes the target board may require some kind of periodic waveform with a particular frequency as input to some part of the board.
- Thus, in a debugging environment, the function generator serves the purpose of generating and supplying required signals.