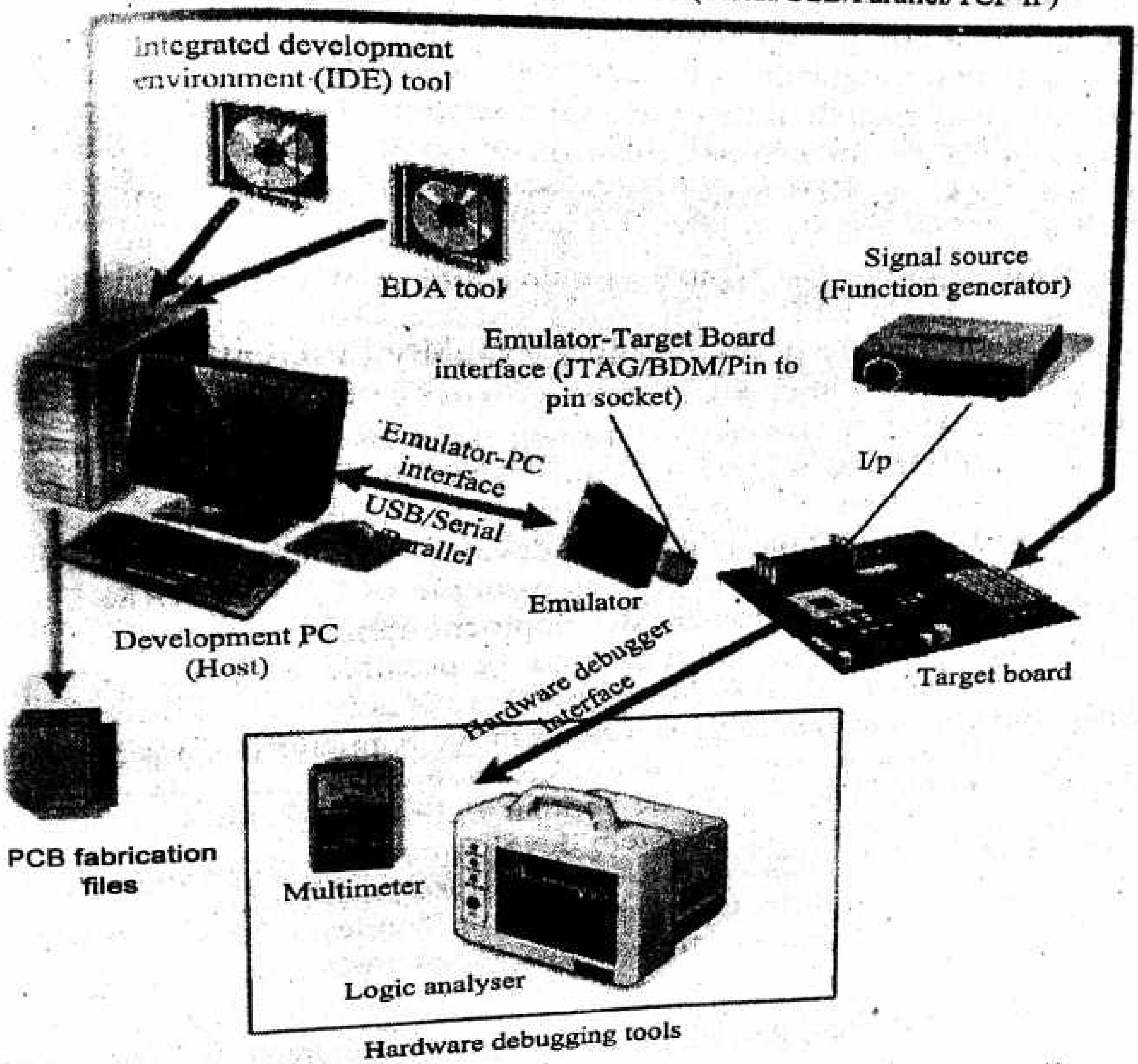
THE INTEGRATED DEVELOPMENT ENVIRONMENT (IDE)

In embedded system development context, Integrated Development Environment (IDE) nds for an integrated environment for developing and debugging the target processor specific bedded firmware. IDE is a software package which bundles a 'Text Editor (Source Code itor)', 'Cross-compiler (for cross platform development and compiler for same platform velopment)', 'Linker' and Debugger'. Some IDEs may provide interface to target board nulators, Target processor's /controller's Flash memory programmer, etc. and incorporate other ftware development utilities like 'Version Control Tool', 'Help File for the Development anguage', etc. IDEs can be either command line based or GUI based. Command line based DEs may include little or less GUI support. The old version of TURBO CIDE for developing pplications in C/C++ for x86 processor on Windows platform is an example for a generic IDE vith command line interface. GUI based IDEs provide a Visual Development Environment with nouse click support for each action. Such IDEs are generally known as Visual IDEs. Visual IDEs are very helpful in firmware development. A typical example for a Visual IDE is Microsoft® Visual Studio for developing Visual C++ and Visual Basic programs. Other examples are Net Beans and Eclipse.

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. MPLAB is an IDE tool supplied by microchip for developing embedded firmware using their PIC family of microcontrollers. Keil µVision3 (spelt as micro vision three) from Keil software is an example for a third party IDE, which is used for developing embedded firmware for 8051 family microcontrollers. Code Warrior by Metrowerks is an example of IDE for ARM family of processors. It should be noted that 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 IDE (as of now there is no known IDE with support for all



family of processors/controllers). However there is a rapid move happening towards the ope source IDE, Eclipse for embedded development. Most of the processor/control manufacturers ar third party IDE providers are trying to build the IDE around the popular Eclipse open sourc IDE. This may lead to a single IDE based on Eclipse for embedded system development in th near future. Since this book is primarily focusing on 8051 based embedded firmwar development, the IDE chosen for demonstration is Keil µ Vision3. A demo version of the tool fo Microsoft Windows OS based development is available for free download from the Kei Software website. Please install the same on your machine before proceeding to the next sections.

2 TYPES OF FILES GENERATED ON CROSS-COMPILATION

Cross-Compilation is the process of converting a source code written in high level language (like 'Embedded C') to a target process or controller understandable machine code (ex: ARM processor or 8051 microcontroller specific machine code). The conversion of the code is done by software running on a processor / controller (ex: x86 processor based pc) which is different from the target processor. The software performing this operation is referred as the 'Cross-compiler'. Cross assembling is similar to Cross-compiling; the only difference is that the code written in a target processor / controller specific Assembly code is converted into its corresponding machine code. The application converting Assembling 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 generated various types of intermediate files. Various files generated during the cross-compilation / cross-assembling process are: List File (.lst), Hex File(.hex), pre-processor out put file, Map file (File extension linker dependent), Object file (.obj).

List File (.lst file):

Listing file is generated during the cross-compilation process and is contains an abundance of information about the cross compilation process, like cross compiler details, formatted source text ('C' code), assembly code generated from the source file, symbol tables, errors and warnings detected during the cross-compilation process. The type of information contained in the list file is cross-compiler specific.

Source Code:

The Source code listing outputs the line number as well as the source code on that line. Specific cross compiler directive can be used to include or exclude the conditional codes (code in # if blocks) in the source code listings.

```
Void main ( )
{
Printf ("Hello world!\n)
}
```

Assembly listing:

Assembly listing contains the assembly code generated by the cross compiler for the 'C' source code. Assembly code generated can be excluded from the list file by using special compiler directives.

```
ASSEMBLY LISTING OF GENERATED OBJECT CODE
; FUNCTION main (BEGIN)
        ; SOURCE LINE #5
        ; SOURCE LINE #6
```

```
                    ; SOURCE LINE #7
00007 BFF           MOV R3, #OFF H
0002 7A00           R       MOVR2, # HIGH? SC-0
0004 7900           R       MOVR1, # LOW? SC-0
               ; FUNTCTION main (END)
```

Preprocessor Output file:

The Preprocessor output file generated during Cross-compilatio contain the preprocessor output for the preprocessor instructions used in the source file Preprocessor output file is used for verifying the operation of macros and conditiona preprocessor directives. The preprocessor output file is a valid C source file. File extension o preprocessor output file is cross complier dependent.

Objective File (.OBJ File):

Cross-compiling / assembling each source module (written in C Assembly) converts the various Embedded C / Assembly instructions and other directives present in the module to an object (.OBJ) file. The format (internal representation) of the .OBJ file is cross compiler dependent. OMF51 or OMF2 are the two objects file formats supported by C51 cross compiler. The object file is a specially formatted file with data records for symbolic information, object code, debugging information, library references, 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 synchronies source lines with object code.

Map File (.MAP):

The cross-compiler converts each source code module into a re-locatab object (OBJ) file. Cross-compiling each source code module generates its own list file. In project with multiple source files the cross-compilation of each module generates corresponding object file. The object files so created are re locatable codes, meaning thei location in the code memory is not fixed. It is the responsibility of a linker to link all these objec files. The locater is responsible for locating 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.

HEX File (.HEX):

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. Hex files embed the machine code in particular format. 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. Each record is made up of hex a decimal 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 programming and data to a ROM or EPROM which is used as code memory storage.

7.3 DISASSEMBLER / DECOMPILER

Disassemble 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 / cross-assembling. De compiler is the utility program for translating machine cod into corresponding high level language instructions. De compiler performs the reverse operatic of compiler / cross-compiler. The dis-assemblers / de compilers for different family of processo / controllers are different. Disassemblers / de compliers are deployed in reverse engineerin Reverse engineering is the process of revealing the technology behind the working of a produc Reverse engineering in embedded product development is employed to find out the secret behir the working of popular proprietary products. Disassemblers / decompilers help the revers engineering process by translating the embedded firm ware into Assembly / high level languag instructions.

Disassemblers / decompilers are powerful tools for analyzing the presence o malicious codes (virus information) in an executable image. Disassemble/decompilers are available as either free ware tools readily available for free download from internet or as commercial tools. It is not possible for a disassembler / decompiler to generate an exact replica of the original assembly code/high level source code in terms of the symbolic constants and comments used. However disassemble / decompilers generates a source code which is some-what matching to the original source code from which the binary code is generated.

7.4 SIMULATORS, EMULATORS AND DEBUGGING

Simulators and emulators are two important tools used in embedded system development. Both the terms sound a like and are little confusing. Simulator is a software tool used for simulating the various conditions for checking the functionality of the application firmware. The Integrated Development Environment (IDE) itself will be providing simulator support and they help in debugging the firmware for checking its required functionality. In certain scenarios, simulator refers to a soft model (GUI model) of the embedded product. For ex. It the product can be developed in software. Soft phone is an example for such a simulator.

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.

7.4.1 SIMULATORS

Simulators simulate the target hardware and the firmware execution can be inspected using simulators. The features of simulator based debugging are listed below.

1) Purely software based
2) Doesn't require a real target system
3) Very primitive (Lack of featured I/O support. Everything is simulated one).
4) Lack of real-time behavior.

Advantages of Simulator Based Debugging:

Simulator based debugging techniques are simple and straight forward. The major advantages of simulator based firmware debugging techniques are explained below.

1) No Need for original Target Board

Simulator based debugging technique is purely soft ware oriented. IDE's software support simulates the CPU of the target ward. User only needs to know about the memory map of various devices within the target board, and the firmware should be written on the basis of it. Since the real hardware is not required, firmware development can start well in advance immediately after the device interface and memory maps are finalized. This saves development time.

2) Simulate I/O Peripherals

Simulator provides the option to simulate various peripherals. Using simulator's I/O support you can edit the values for I/O registers and can used as the input / output value in the firmware execution. Hence it eliminates the need connecting I/O devices for debugging the firmware.

3) Simulates Abnormal Conditions

With simulator's simulation support you can input a desired value for any parameter during debugging the firmware and can observe the control fl of firmware. It really helps the developer in simulating abnormal operational environment firmware and helps the firmware developer to study the behavior of the firmware under abnorm input conditions.

Limitations of simulator based Debugging

Though simulation based firmware debuggin technique is very helpful in embedded applications. Some of the limitations of simulator base debugging are explained below.

→ Deviation from Real Behaviour

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 timelines

The major limitation of simulator based debugging is that it is not real-time in behavior. The debugging is developer driven and it is no way capable of creating a real time behavior. Moreover in a real application the I/O condition may be varying or unpredictable. Simulation goes for simulating those conditions for known values.

7.4.2 EMULATORS AND DEBUGGERS

Debugging process in embedded application is broadly 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. Firm ware 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.

Firm ware debugging is preformed to figure the bug or the error in the firmware which creates the unexpected behavior. Firmware is analogous to the human body in the sense it is wide spread and / or modular. During the early days of embedded system development, there were no debug tools available and only ways was "Burn the code in an EEPROM".

The following section describes the improvements over firmware debugging starting from the most primitive type of debugging to the most sophisticated on chip debugging (OCD).

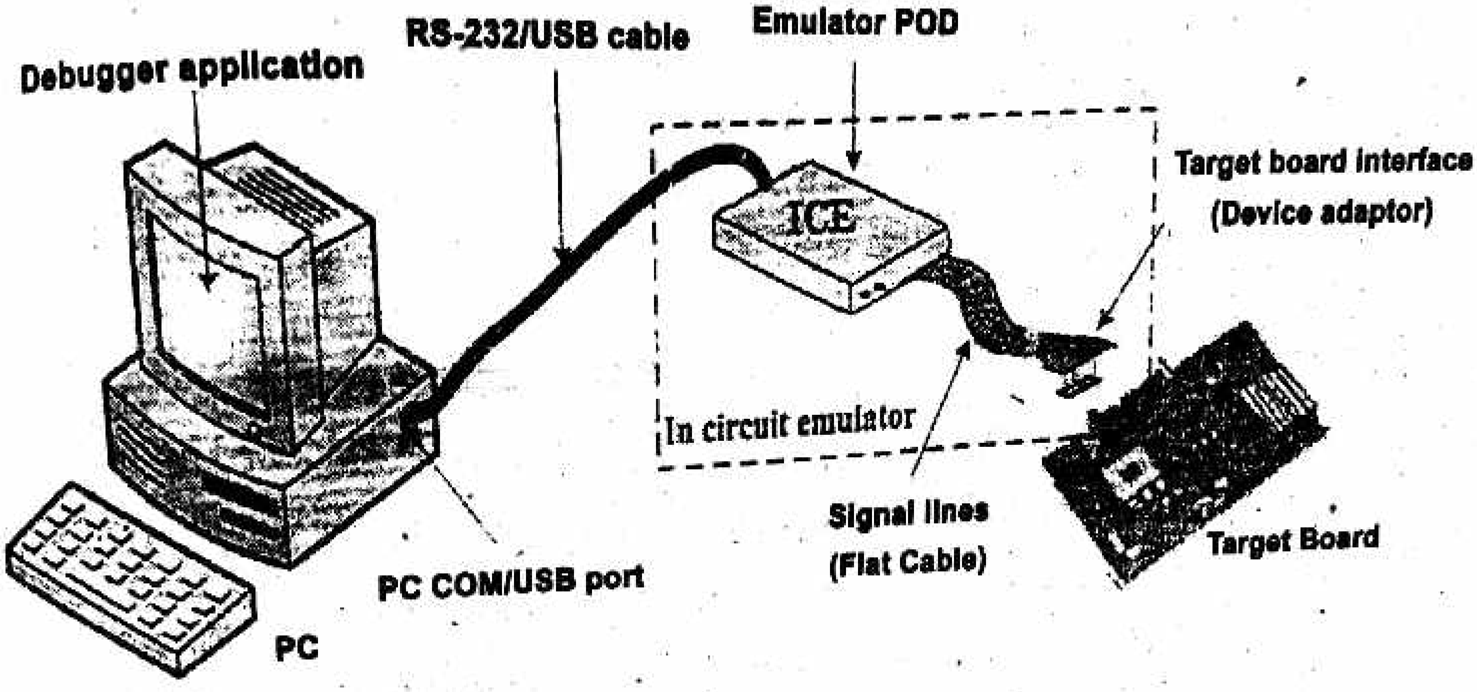


Fig.7.3

.5 EMULATION DEVICE

Emulation device is a replica of the target CPU which receives rious signals from the target board through a device adaptor connected to the target board and rforms the execution of firmware under the control of debug commands from the debug plication. The emulation device can be either a standard chip same as the target processor (ex. T89C51) or a programmable logic device(PLD) configured to function as the target CPU.

mulation 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'.

Emulator Control Logic:

Emulator control logic is the logic circuits used f implementing complex hardware break points, trace butter, trigger detection, trace buff controller, etc.. Emulator control logic circuits are also end for implementing logic analyz functions in advanced emulator devices.

Device Adaptors:

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

7.4.6 ON CHIP FIRMWARE DEBUGGING(OCD)

Advances in semiconductor technology has brought out new dimensior to target firmware debugging. Today almost all processors / controllers in corporate built i debug modules could On Chip Debug(OCD) support. Though OCD adds silicon complexity an cost factor, from a developer perspective it is a very good feature supporting fast and efficier firmware debugging.

Chips with JTAG debug interface contain a built-in JTAG port fo communicating with the remote debugger application. JTAG is the acronym for Joint Test Actio Group. JTAG is the alternate name for IEEE 1149.1 standard. Like BDM, JTAG is also a seria interface. The signal lines of JTAG protocol are explained below.

Test Data In (TDI):

It is used for sending debug commands serially from remote debugger to the target processor.

Test Data Out (TDO):

Transmit debug response to the remote debugger from target CPU.

Test Clock (TCK):

Synchronizes the serial data transfer.

Test Mode Select (TMS):

Sets the mode of testing.

Test Reset (TRST):

It is an optional signal line used for resetting the target CPU.

The serial data transfer rate for JTAG debugging is chip dependent. It is usually within the range of 10 to 1000 MHz.

7.5 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 explained below.

Magnifying Glass (Lens):

Magnifying glass is the primary hardware debugging tool fo embedded hardware debugging professional. A magnifying glass is a powerful visual inspe tool. With a magnifying glass (lens), the surface of the target board can be examined thorou for dry soldering track (PCB connection) damage, short of tracks, etc.. Now a days high qu magnifying stations are available for visual inspection. The magnifying station incorpo magnifying glasses attached to a stand with CFL tubes for providing proper illumination inspection. The statin usually incorporates multiple magnifying lenses.

Multi meter:

Multi meter is used for measuring various electrical quantities like voltage (E Ac and DC), current (DC as well as AC) resistance, capacitance, continuity checking, trans checking, cathode and mode identification of diode. In embedded hardware debugging mainly used for checking the circuit continuity between different points on the board meas the supply voltage, checking the signal value, polarity etc. the digital version is preferred analog the one for various reasons like readability, accuracy, etc..

Digital CRO:

CRO is a little more sophisticated tool compared to a multimeter. CRO is use wave from capturing and analysis, measurement of signal strength. CRO is very good analyzing interference noise in the power supply line and other signal lines. Monitoring crystal oscillator signal from target board is a typical example of the usage of CRO for wave capturing and analysis in target board is a typical example of the usage of CRO for ware capturing and analysis in target board debugging. CRO's are available in both analog and di versions. Though digital CRO are costly and are best suited for forget board debug applications. Modern digital CRO's more than one channel and it is easy to capture and anal various signals channel and it is easy to capture and analyze various signals from the target b using multiple channels simultaneously.

Logic Analyzer:

A logic analyzer is the big brother of digital CRO Logic analy for capturing digital data (logic 1 and 0) from a digital circuitry where as CRO is en capturing all kinds of waves including logic signals. Another major limitation of CRO total number of logic signals / wave forms that can be captured with a CRO is a limit number of channels. A logic analyzer contains special connectors and clips which can be to the target board for capturing digital data. In target board debugging applications analyzer captures the states of various port pins, address bus and data bus to the target pro controller. Most modern logic analyzers contain provisions for storing captured data sele desired region of the captured ware form, zooming selected region of the captured ware for

Function Generator:

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

7.6 BOUNDARY SCAN

The device packages used in the PCB become miniature to reduce th total board space occupied by them and multiple layers may be required to route the inte connections among the chips with miniature device packages and multiple layers for the PC i will be very difficult to debug the hardware using magnifying glass, multi meter to check the interconnection among the various chips. Boundary scan is a technique used for testing the ... the various chips, which support JTAG interface, present in the board, chips which support boundary scan associate a boundary scan cell with each pin of the device. A JTAG port which contains the five signal lines namely TDI, TDO, TCK, TRST and TMS form the Test Access Port (TAP) for a JTAG supported chip. Each device will have its own TAP. The PCB also contains a TAP for connecting the JTAG signal lines to the external world. A boundary scan path is formed inside the board by inter connecting the devices through JTAG signal lines. The TDI pin of the TAP of the PCB is connected to the TDI pin of the first device. The TDO pin of the first device is connected to the TDI pin of the 2nd device. In this way all devices are interconnected and the TBO pin of the last JTAG device is connected to the TBO pin of the TAP of the PCB.