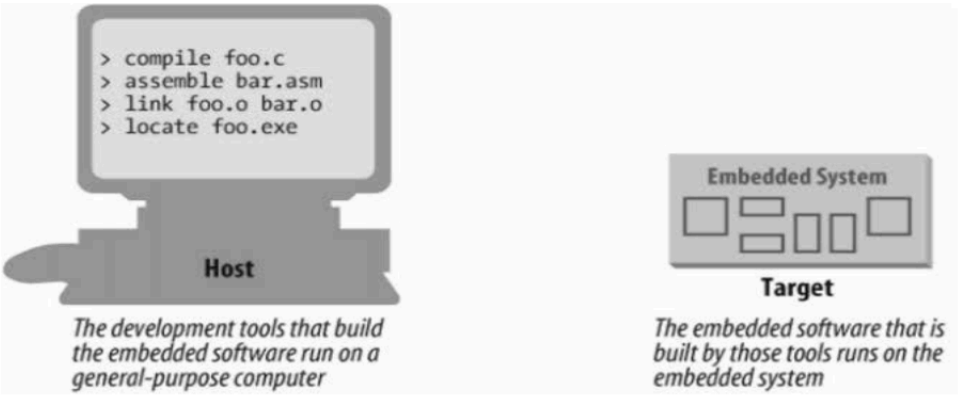
CO5 VALIDATION AND DEBUGGING

Host and Target Systems:

- **Host - generally PC or laptop or workstation.**
  The host system refers to the computer or computing device where the software development tools, compilers, debuggers, and other necessary software components are installed and executed. It is typically a PC, laptop, or workstation that developers use to write, compile, and test software intended for deployment on another platform, such as an embedded system or server.

- **Target - actual hardware for the embedded system under development.**
  The target system refers to the actual hardware or platform for which the software is being developed. In the context of embedded systems development, the target system is the embedded device or hardware platform where the software will ultimately run. Developers typically cross-compile the software on the host system to generate executable code that is compatible with the target hardware. The target system may include microcontrollers, microprocessors, sensors, actuators, and other hardware components tailored for specific applications.



Host and Target Testing:

- Test at initial stages are at the host.
- Host is used to test hardware independent codes.
- Host is also used to run simulator.
- Target is used to test hardware dependent codes.

Target Testing:

- Definition: Testing performed on actual embedded systems or very similar replicas under real-time conditions.
- Example: Testing a TV remote control by connecting it to a radio and performing sanity checks.
- Advantages: Comprehensive; detects errors quickly and in larger numbers.

Host Testing:
The software of embedded systems is detached from its surroundings and it can be tested in a limited way on a host machine by simulation of all neighboring environments.

HOST TESTING METHODS

- Manual testing
- Automated testing
- Regression testing
- Whitebox testing
- Functional testing

Host-based testing setup

- The concept of host-based testing involves a simulation of the complete environment of the system under test (SUT).
- Since the hardware dependent portions of the code cannot be tested effectively on host, they need to be simulated as well.
- There is a part of code inside the system that is completely hardware independent, and a piece of code that depends on the hardware.
- The idea is to replace this hardware dependent code with a test setup that simulates this environment.
- The simulation test code needs to be written for ISRs, timing interrupts, direct access to memory and devices, other modules, etc. Usually, such test code is called a “stub”.
- The software of embedded systems detached from its surroundings and it can be tested in a limited way on a host machine by simulation of all neighboring environment

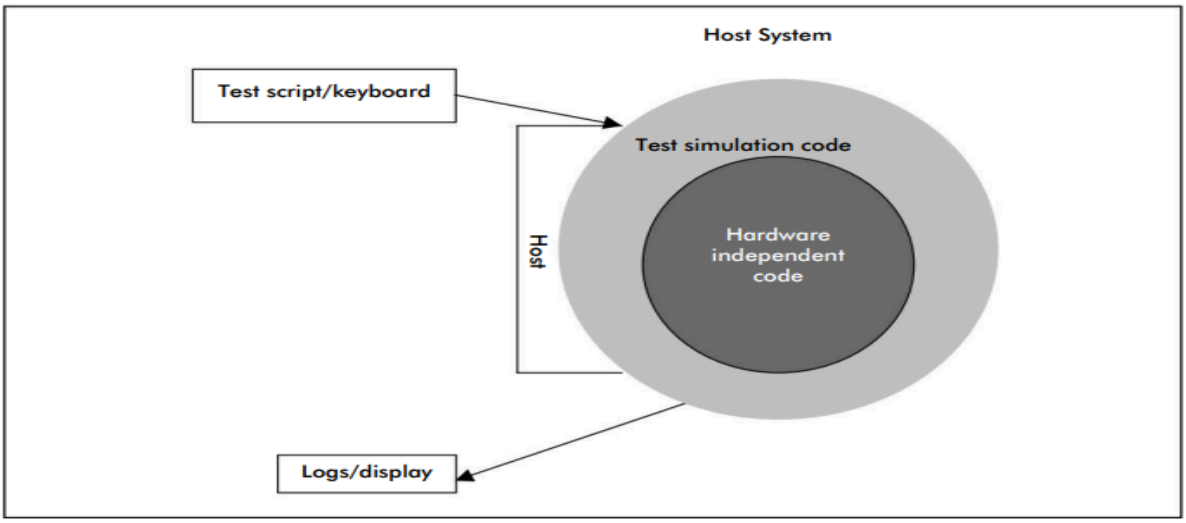


Fig. 12.1 *Host test setup*

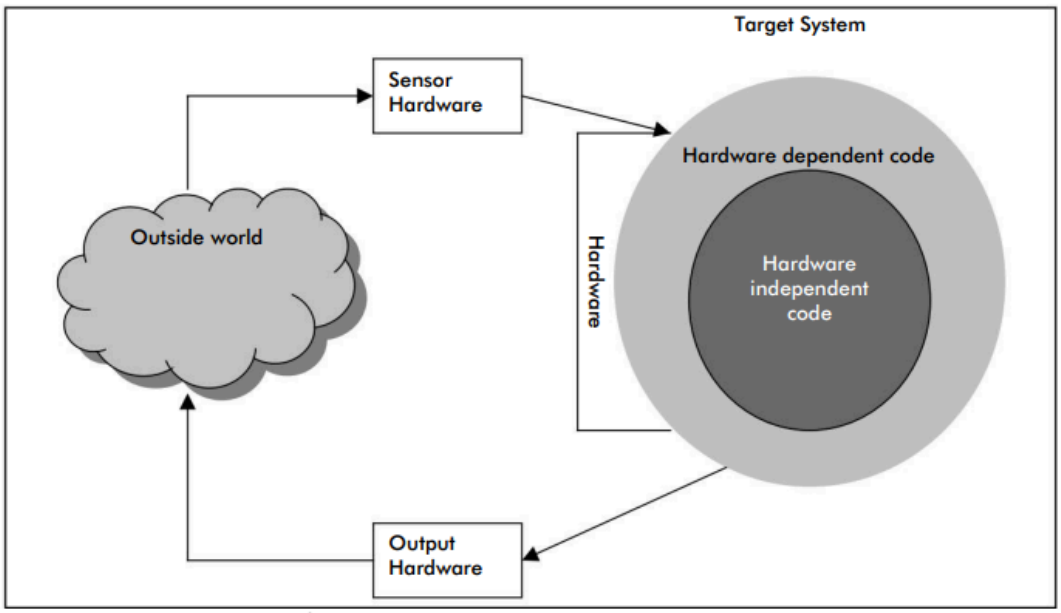


Fig. 12.2 *The organisation of target system*

Manual Testing :

- Manual testing involves human testers executing test cases without automated tools.
- It's useful for quick feedback on specific test cases but requires less planning and is prone to human error.
- Manual testing of embedded systems is most useful in situations where the results of a specific and limited set of test cases are needed relatively quickly.

Advantages of Manual Testing:

- Quick feedback: Manual testing provides rapid assessment and decision-making by offering immediate feedback on tested components or functionalities.
- Flexibility: Human testers can adapt and explore the system, identifying nuanced issues that automated tests may overlook.
- Exploration: Manual testing allows for exploratory testing, uncovering unexpected behaviors or edge cases that may not be covered by predefined test cases.

Automated Testing:

- Automated testing utilizes tools and scripts to execute test cases.
- It offers benefits like reduced test time and improved quality, though it requires more initial effort. Automation enables consistent testing and accumulates tests over time for regression testing.

Advantages of Automated Testing:

- Reduced test time: Automated testing significantly reduces the time required to execute test cases, allowing for faster verification of system behavior.
- Increased test coverage: Automation enables the execution of a larger number of test cases, resulting in improved test coverage and earlier detection of defects.
- Repeatability: Automated tests produce consistent and repeatable results, enabling reliable regression testing and comparison of system behavior across different builds or versions.

Numerical Example:

- Consider the scenario of developing a target system that accepts n different types of input values, ranging from 1 to n, and generates an encryption key based on these values. This means: values of 0, 1, 2, 100/2, 100-1, 100 and 100+1
- For n=4, there would be 777*7 = 2401 possible input combinations to test.
- Manual testing of all these combinations would be impractical, taking over 40 hours to complete, even under the best conditions.
- Alternatively, automating this test drastically reduces execution time, allowing testers to allocate their time more efficiently. After initial setup, the tester can focus on performing manual testing for additional coverage.
- If, for example, 80% of the test plan can be automated, only 4 days would be spent on automated testing, achieving results comparable to 20 days of manual testing. This allows the remaining time to be utilized for manual testing, potentially uncovering additional bugs that may have been missed otherwise.

Difference

Manual Testing	Automated Testing
- Involves human testers executing test cases	- Utilizes automated tools or scripts to execute test cases
- Provides relatively quick feedback on specific	- Requires initial effort to plan, organize, and produce tests
and limited test cases	- Produces repeatable tests that can be run in batches and logged automatically
- More suitable for scenarios where quick results	- Offers advantages in terms of reduced test time and staff effort
are needed	- Accumulates tests over time for regression testing and quality monitoring
- Becomes increasingly challenging and time-	- Requires an initial investment of time but facilitates earlier defect detection
consuming as systems grow in complexity	and correction through extensive test coverage
- Offers flexibility for exploration and human	- Enhances overall quality of the product by running a greater number of tests
intuition in uncovering critical issues	

Regression testing:

- Regression testing is used to test previously observed bugs in the code. This testing is best performed in an automated manner whenever a new baseline for the software is created or after a bug has been fixed. The purpose of regression testing is twofold:

❑ Sanity check: Regression testing performs effective sanity checks for the system. After going through a regression testing phase, the minimum quality of the software is guaranteed such that basic minimum paths and operations have been tested.
❑ Old ghosts: It is common for software developers to introduce new bugs while fixing a problem, or to fix the problem partially in the first place. Regression testing makes it possible to detect any old ghosts returning.

- Regression test is more to control re-occurrence of past defects. Hence, they can be created only after faults have been detected in the system.

- Alternatively, all tests created for a system in the beginning can renegade to the status of being regression tests later in the next versions.
- Usually, regressions tests are written on the basis of reports from the field, or from a past error. Unless, a very remote and complicated path of software has been found to be faulty, software bugs usually exist in groups. If a portion of software has been found to be faulty, chances are that more problems can be unearthed by changing the parameters slightly and checking other border conditions.

White Box Testing:
- Definition:
White box testing is performed at the system or module level by a team to exercise the most important paths of the source code.

- Scope:
Tests coverage of the code, focusing on how the code has been written rather than verifying if it matches the requirements.

- Dependencies:
The tester needs to understand the organization and functionality of the code.

- Timing:
Typically conducted after developers have completed unit testing.

- Limitations:
  - May not detect missing code faults.
  - Some parts of the code may remain unreachable during testing.
  - Heavily dependent on the structure and quality of the code.
  - Susceptible to changes in code due to evolving requirements or bug fixes.

- Collaboration:
Difficult to perform in isolation from the development team, as testers need to understand the code and collaborate closely with developers.

- Inadequate for:
Testing system-level issues such as timing requirements, hardware interfaces, and load or stress testing.

Functional/Black box testing:

- Definition:
 Functional testing, also known as black box testing, assesses the system based on its requirements to determine if it meets expected criteria.

- Scope:
 Focuses solely on whether the system satisfies the current requirements, without consideration for code coverage or past regression.

- Independence:
Independent of system design and implementation, allowing for parallel development and testing activities.

- Suitability for Nonfunctional Requirements:
 Well-suited for testing nonfunctional requirements due to its requirement-centric approach.

- Strengths:
  - Can identify problems not detectable by regression or code-based testing.
  - Allows for early development of test suites parallel to source code development.

- Dependencies on Requirements Quality:
  - Effectiveness heavily depends on the quality and stability of the requirements.
  - Rapid changes in specifications can limit the effectiveness of functional testing.

- Limitations:
  - Does not analyze code coverage as its primary focus is on requirements.
  - Ineffective as a substitute for code-based testing, which is more cost-effective for code coverage analysis.

Limitations of Host Testing:

1. No Real-time Testing:
   - Difficult to test problems related to real-time behavior on the host, even with simulators.

2. Peripheral and Memory Access:
   - Shared data problems and issues with peripherals are hard to detect on the host.
   - Problems like temperature increases or spurious interrupts are easier to detect on the target.

Target Testing Types:

ROM Emulator:

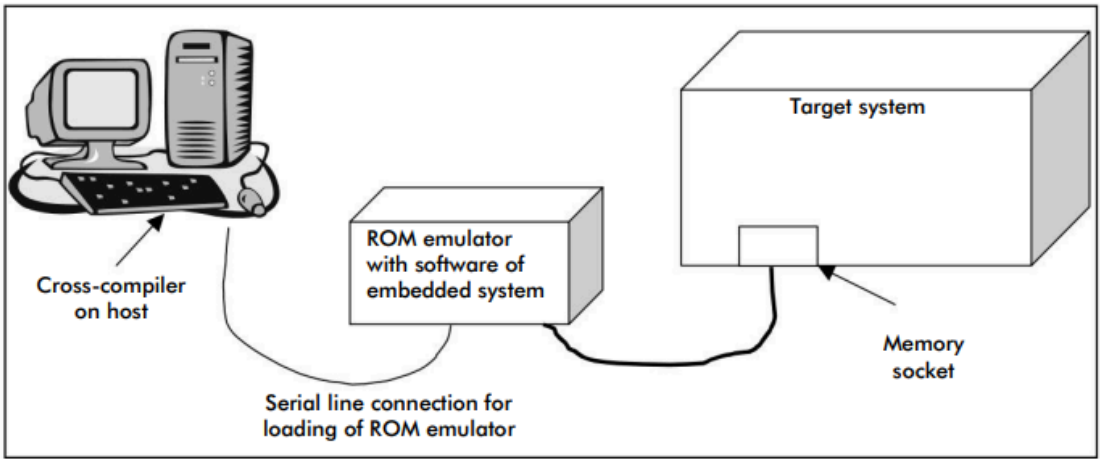


Fig. 12.3 *Setup for ROM emulator*

- A ROM emulator is an electronic circuit with interfaces for both the host and target systems, serving as an alternative to programmable read-only memory (PROM).
- The emulator is loaded with a cross-compiled image and connects to the target system via a bus, replacing the need for a physical ROM.
- Simplifies code changes as new code can be compiled and loaded onto the emulator without the need for physical reprogramming.
- Offers flexibility and convenience in updating code, eliminating the need for frequent PROM programming.
- Functions as a direct replacement for ROM, providing seamless integration into the target system's memory socket.
- It allows quick download of new object code images to run in the target system

The ROM emulator contains the following system elements:
f Cabling device(s) to match the target system mechanical footprint of
the target system ROM devices
f Fast RAM to substitute for the ROM in the target system
f Local control processor
f Communications port(s) to the host
f Additional features, such as trace memory and flash programming algorithms

Table 6.2: Advantages/disadvantages of ROM emulator.

Advantages of the ROM emulator		Disadvantages of the ROM emulator
▪ Very cost-effective ($1,000– $5,000) ▪ Generic tool, compatible with many different memory configurations ▪ Can download large blocks of code to the target system at high- speed ▪ Most cost-effective way to support large amounts of RAM substitution memory ▪ Can trace ROM code activity in real time ▪ Provides virtual UART function, eliminating need for additional services in target system ▪ Can be integrated with other hardware and software tools, such as commercially available debuggers ▪ Can set breakpoints in ROM		▪ Requires that the target system memory is in a stable condition ▪ Feasible only if embedded code is contained in standard ROMs, rather than custom ASICs or microcontrollers with on-chip ROM ▪ Real-time trace is possible only if program executes directly out of ROM ▪ Many targets transfer code to RAM for performance reasons

Remote Debuggers:

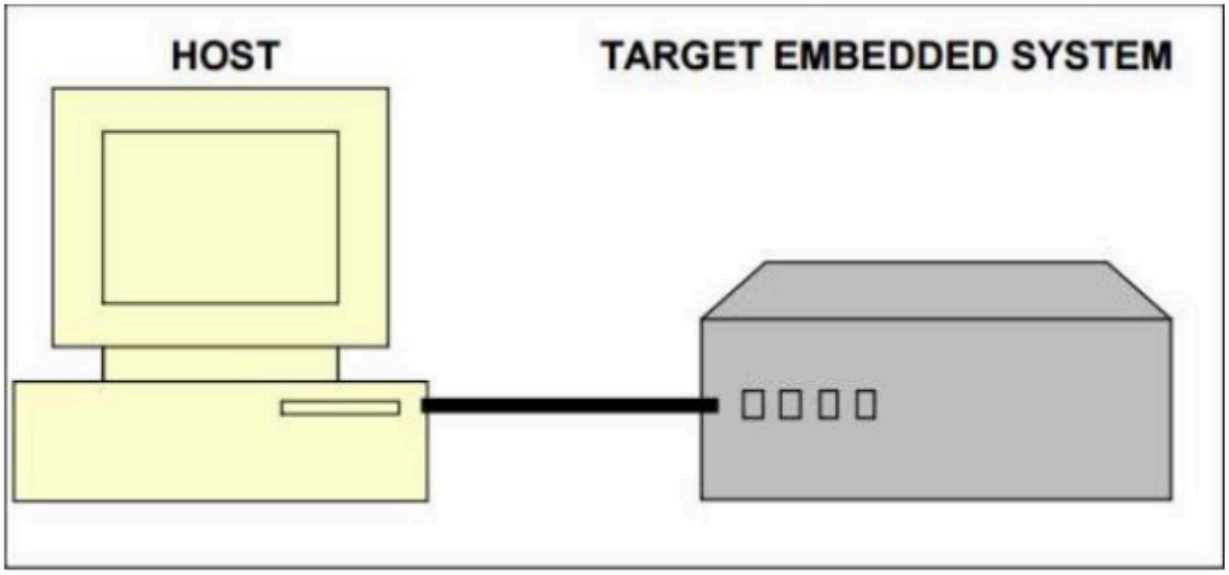
A remote debugger is a software tool used in software development and debugging processes, designed to facilitate the identification and resolution of issues in a target system from a remote location. It consists of two main components:

Frontend Remote Debugger:

- Definition: The frontend remote debugger operates on a host computer and serves as the user interface, enabling developers to interact with the debugging tools remotely.
- Functionality: It provides a graphical or command-line interface through which developers can control the debugging session, set breakpoints, inspect variables, and analyze runtime information.
- Communication: Establishes communication with the backend remote debugger running on the target processor to send commands, receive data, and synchronize debugging actions.

Backend Remote Debugger:

- Definition: The backend remote debugger operates on the target processor or embedded system itself, executing debugging operations within the target environment.
- Functionality: It manages the execution of debugging commands received from the frontend debugger, controls the debugging process at the target level, and provides real-time insights into the behavior of the embedded system.
- Communication: Establishes a communication link with the frontend debugger over a network connection or serial interface, enabling bidirectional data exchange and command execution.



Debug Kernels:

The debug kernel requires two resources from the target. One is an interrupt vector, and the other is a software interrupt.The interrupt vector for the serial port (assuming that this is the communications link to the host) forces the processor into the serial port ISR, which also becomes the entry point into the debugger. Again, this assumes that the serial port's interrupt request will be taken by the target processor most, if not all, of the time. After the debug kernel is entered, the designer is in control of the system. The debug kernel controls whether other lower-priority interrupts are accepted while the debugger is in active control. In many situations, the target system crash as if the debugger does not re-enable interrupts.

(In simpler terms, when debugging embedded systems, a special piece of software called the "debug kernel" is used. This debug kernel needs two things from the target system:

1. Interrupt Vector This is like a signal that tells the processor to stop what it's doing and handle a particular task. In this case, the interrupt vector is used for the serial port, which is often the connection to the computer running the debugger. When the serial port receives data, it triggers an interrupt that brings the processor's attention to the debugger.

2. Software Interrupt: This is a command sent by software to request a specific action from the processor. In this context, it's used as a way for the debugger to communicate with the target system.

Once the debugger is activated, the designer has control over the system. The debugger can decide whether to allow other less important tasks to interrupt its debugging process. However, if the debugger doesn't properly handle these interruptions, it can sometimes cause the target system to crash or freeze.
)

• The portion of the debugger that resides in the target is called the target agent or the debug kernel.
•It implements the "Run Control" function
- Examine/modify memory or registers
- Single step
- Run to breakpoint
- Load code are implemented

Table 6.1: Advantages/disadvantages of the debug kernel.

Advantages of the debug kernel		Disadvantages of the debug kernel
Low cost: $0 to < $1,000 Same debugger can be used with remote kernel or on host Provides most of the services that software designer needs Simple serial link is all that is required Can be used with "virtual" serial port		Depends on a stable memory sub system in the target and is not suit able for initial hardware/software integration Not real time, so system performance will differ with a debugger present Difficulty in running out of ROM- based memory because you can't sin gle step or insert breakpoints Requires that the target has addi tional services, which, for many tar get systems, is not possible to implement Debugger might not always have control of the system and depends on code being "well behaved"
Can be linked with user's code for ISRs and field service Good choice for code development when hardware is stable Can easily be integrated into a design team environmen		

Logic Analysers

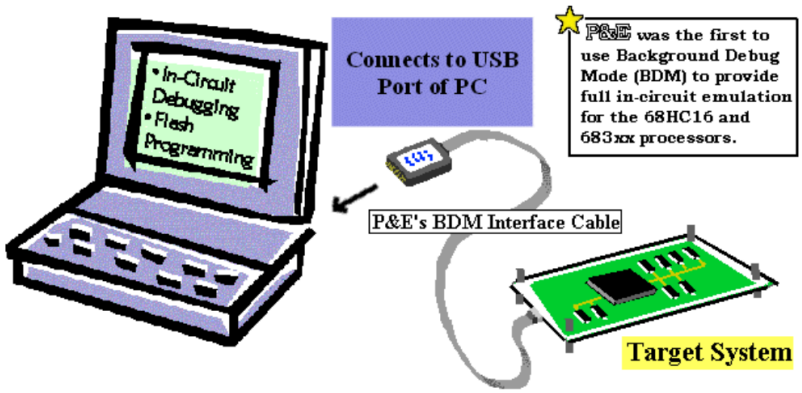
- They are used to check the logical level of input pins in realtime.
- Usually it is possible to connect a number of inputs pins for smart analysis by programming the logical analyser: Start tracing pins C and D when the inputs on pins A and B are 1.
- Usually, logic analysers are used to debug hardware circuits in conjunction with other methods
- described in this section.
- Logic analysers in a sense are a smarter version of oscilloscopes with a flexible event system. They also have displays showing different data values observed on different pins as programmed by the user.
- The logic analysers are actually specialised oscilloscopes for embedded systems, but unlike oscilloscopes, can measure and report voltages as either logical high or low, nothing in between.

- This limitation, however, suits most embedded systems perfectly since they anyway measure voltages in that way.

BDM

BACKGROUND DEBUG MODE

How do modify firmware when it's embedded on a chip where a ROM emulator can't be used?



- BDM (Background Debug Mode) serves as Motorola's proprietary debug interface tailored for embedded systems, pioneering the integration of specialized debugging circuitry within the processor core itself.
- It introduces on-chip debugging resources, a now widespread trend among embedded processors, streamlining debugging processes without relying on costly external tools.
- A key element of BDM is the "wiggler," an interface module that connects to the target system's debug port, manipulating several processor pins to implement the debug core's communication protocol.
- Compared to traditional in-circuit emulators (ICE), wigglers are considerably more cost-effective, democratizing debugging accessibility for developers.
- Initially deployed with the 683XX family, BDM is commonly utilized with the ColdFire processor family, interfacing through a 26-pin connector on the target PC board.
- Noteworthy within BDM's capabilities is its support for both processor control and real-time trace monitoring, delivering debug data and processor status updates while the processor operates at full speed.
- BDM transmits processor status codes via pins DDATA0-DDATA3 and PST0-PST3, enabling third-party tools to analyze the processor core's execution flow.
- Special instructions in the ColdFire instruction set, such as PULSE and WDDATA, facilitate the seamless integration of debug core operations with instruction execution flow, facilitating tasks like execution time measurements.
- BDM commands function autonomously from any program code execution, interfacing directly with the CPU core to deliver debugging functionalities.

- Leveraging the ColdFire processor's debug core, real-time debugging is supported by additional registers programmable via the BDM port, enabling breakpoint triggering and execution halting or debug interrupts initiation under specific conditions. This empowers developers to define custom debugging routines while the processor maintains instruction execution continuity.
BDM provides direct access to the processor core and allows for features like examining and modifying memory or registers, single-stepping through code, running to breakpoints, and loading code. These capabilities suggest that BDM could potentially facilitate firmware modification on embedded chips where a ROM emulator is not feasible.

PROM programmer
- Directly burning the software image into a ROM for testing on the target is not advisable due to the difficulties and costs associated with making modifications.
- PROMs (Programmable Read-Only Memory) offer a viable alternative to ROMs, especially when the software is not yet stable or requires frequent changes.
- PROM programmers are software tools designed to load software images into PROMs, simplifying the process of creating the image inside the PROM.
- When changes are needed in the software, there are two options:
  1. If the PROM is E-PROM (Erasable Programmable Read-Only Memory), it can be placed inside an eraser and reprogrammed with the new software.
  2. If the PROM is not erasable, it needs to be replaced entirely with a new PROM that is programmed with the updated software.
- Creating a compatible version of the image for the (E) PROM often requires an ad hoc approach to ensure proper functionality and compatibility.

Source level Debuggers

- Source-level debugger, also known as a debug monitor, is a fundamental debugging tool for target systems.
- It involves connecting a host machine with the target machine, with the debugger running on the host providing user interaction and display features.
- The software on the target loads the image into RAM and executes it, allowing for debugging directly on the target system.
- Advantages of source-level debuggers include their affordability compared to advanced tools like emulators.
- Testing and debugging can be done more in real-time on the actual target system, using familiar debugging interfaces.
- Source-level debuggers offer features such as creating breakpoints, single-stepping, stopping, and monitoring registers and memory locations.
- However, there are also disadvantages:
  - Source-level debugging is limited to software simulation, restricting access to internal processor details and limiting rigorous breakpoints and tracing.

- The debugger code inserted into the target system is not part of the final product and must be removed after debugging.
- Porting the debug monitor and serial line driver code to the target system is necessary before debugging can commence.

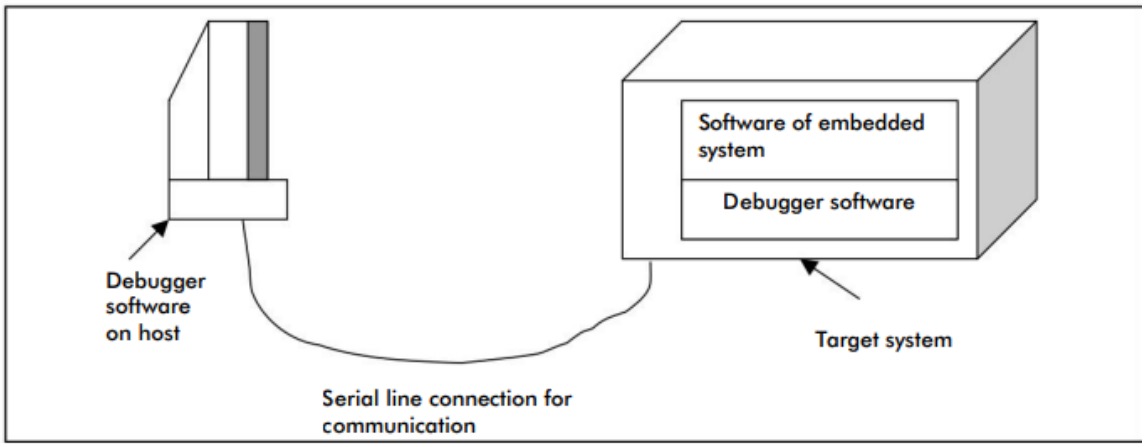
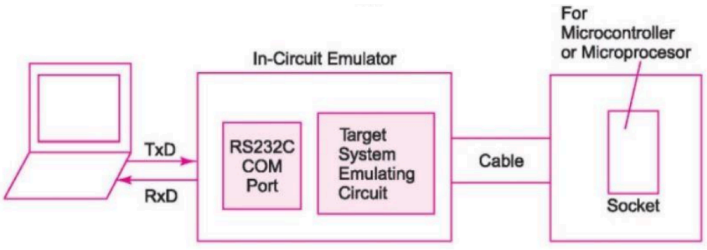


Fig. 12.4 *Setup of source-level debugger*

JTAG

- JTAG is a hardware tool that can control and observe boundary pins of a device for verification of their operation via software control.
- The reason that a special tool had to be created for this purpose is because of the proliferation of number of pins in a given area on a chip.
- A JTAG (Joint Test Action Group) consortium exists that caters to the requirements and standardisation of this testing procedure.
- IEEE 1149.1 standard, known as IEEE Standard Test Access and Boundary Scan Architecture provides complete detail of this procedure.
- For a boundary scan to be possible, the device should be compliant to JTAG, which means that processor provides what is known as a JTAG port.
- A cable connects the host to the JTAG port on the target system and software on the host controls the target microprocessor through it.

Incircuit Emulator:



Bullet-Proof Run Control
Real-Time Trace
Hardware Breakpoints
Overlay Memory

- In-circuit emulator (ICE) is a powerful testing tool for embedded systems, replacing the target processor with electronic test equipment.
- ICE consists of a debugger running on the host system and the emulator running on the target, connected via a bus using UART for information exchange.
- ICE offers a graphical user interface on the debugger, showing execution states, trace messages, and register statuses.
- It allows for setting hardware breakpoints and conditions that are difficult to check using source-level debuggers.
- ICE supports tracing in real-time, capturing processor cycles, timing information, message flows, and contents.
- Information captured by ICE can be transmitted to the debugger or trace window for analysis, including selective tracing capabilities.
- Many ICEs feature overlay memory, enabling them to function as ROM emulators, combining powerful debugging features with the simplicity of loading software patches directly onto the overlay ROM.
- The Trace32-ICE by Lauterbach Corp. is a notable emulator supporting 8 to 32-bit microprocessors with 16 MB emulation memory.

Bullet-Proof Run Control

- In Bullet-Proof Run Control, an In-Circuit Emulator (ICE) uses a debug kernel for run-time control.
- The ICE substitutes its own processor and memory for the target's untested memory/processor interface.
- A cable enables the ICE processor to replace the target's processor, ensuring reliable run control.
- During normal run mode, the ICE reads instructions from the target memory; when needed, it switches to its own local memory for debugging.
- This setup maintains run control even if the target memory is faulty and protects the debug kernel from potential damage by bugs in the target.
- Key components include NMI control logic, memory steering logic, and shadow ROM and RAM.
- When debugging is initiated, the NMI signal is asserted, blocking further NMI-based interrupts and enabling the ICE's local memory to connect to the processor.
- The processor then transitions to the debugger entry point in the shadow ROM for debugging operations.
- This system requires only that the processor has an external NMI capability, ensuring compatibility across different processor architectures.

Real-Time Trace
- Real-Time Trace integration with a generic emulator is straightforward, leveraging existing connections to address, data, and status busses.
- Adding real-time trace involves attaching a logic analyzer to the existing connection, consolidating run control and trace functions into a single target connection.

- This setup allows for controlling the processor and observing its behavior in real-time using the same connection.
- The logic analyzer's trigger signal, used to start and stop the trace, can also be linked to the NMI control logic.
- By connecting the trigger signal to the NMI control logic, the processor can be halted and directed to enter the debug monitor program precisely at the point in the code where the event of interest occurs.
- This integration enhances debugging capabilities by providing synchronized trace data capture and precise triggering for debugging operations.

Hardware Breakpoints

- Hardware breakpoints allow the trigger system of a logic analyzer to take over functionalities previously provided by the debug kernel, enabling real-time breakpoint signaling during program execution.
- Discrete logic analyzers often output trigger pulses that can be used as inputs to BDM or JTAG interfaces to halt execution at specified breakpoints.
- While this method allows for real-time breakpoint setting without slowing down the processor, there might be some execution skew after the breakpoint occurs.
- Real-time trace capability of the logic analyzer provides insights into program behavior, capturing events like infrequent ISR occurrences.
- Trigger signals can be set up based on specific conditions, such as memory address reads or specific events, allowing for targeted debugging.
- Trigger conditions can be tailored to capture states leading up to a failure or exception processing, providing comprehensive insight into system behavior.
- Logic analyzers equipped with symbolic trace information can automate trigger condition setup, eliminating the need for low-level memory details.

```
// Example function foo()
int foo(int bar) {
    int embedded = 15;
    bar++;
    return embedded + bar;
}
```

- `foo()` function returns erroneous values intermittently.
- Suspected stack corruption by an ISR.
- Use a logic analyzer to trigger on `foo()` calls.
- Trigger setup: Instruct the LA to trigger on `foo()` invocation.
- Efficient investigation: LA captures `foo()` execution states for analysis.

Overlay Memory

- Overlay Memory for Emulation:Allows for substituting RAM for ROM in the target system, speeding up the debugging cycle.
- Functionality:Overlay memory operates similarly to shadow memory, with steering logic determining memory access between target and emulation memory.
- Advantages over Substitution Memory: Provides greater flexibility and utility compared to substitution memory, enabling dynamic mapping of memory blocks.
- Memory Mapping System:Utilizes a memory-mapping circuit to map blocks of emulation memory to corresponding blocks in the target system's address space.
- Additional Features:Overlay memory allows for assigning personalities to memory blocks, enabling error detection, memory protection, and coverage testing.

Challenges of Target Testing:

1. Incomplete Software:
   - Testing on target hardware requires complete software, limiting testing until development is finished.
   - Testing individual components before target testing is more cost-effective.

2. Incomplete Hardware:
   - Both hardware and software may not be ready simultaneously, causing delays.
   - Testing on the host while waiting for hardware is more practical.

3. Regression:
   - Simulating bugs again on the target platform for regression testing is difficult.
   - Bugs detected can be effectively used later in a standard test suite.

4. Incomplete Testing:
   - Target testing cannot cover all code portions, especially those related to rare failures.
   - Simulating situations on the host is easier for testing such code.

5. Tracking Methods:
   - Real-time embedded systems lack storage mechanisms for tracking purposes.
   - Special hardware and software are needed for effective tracking.