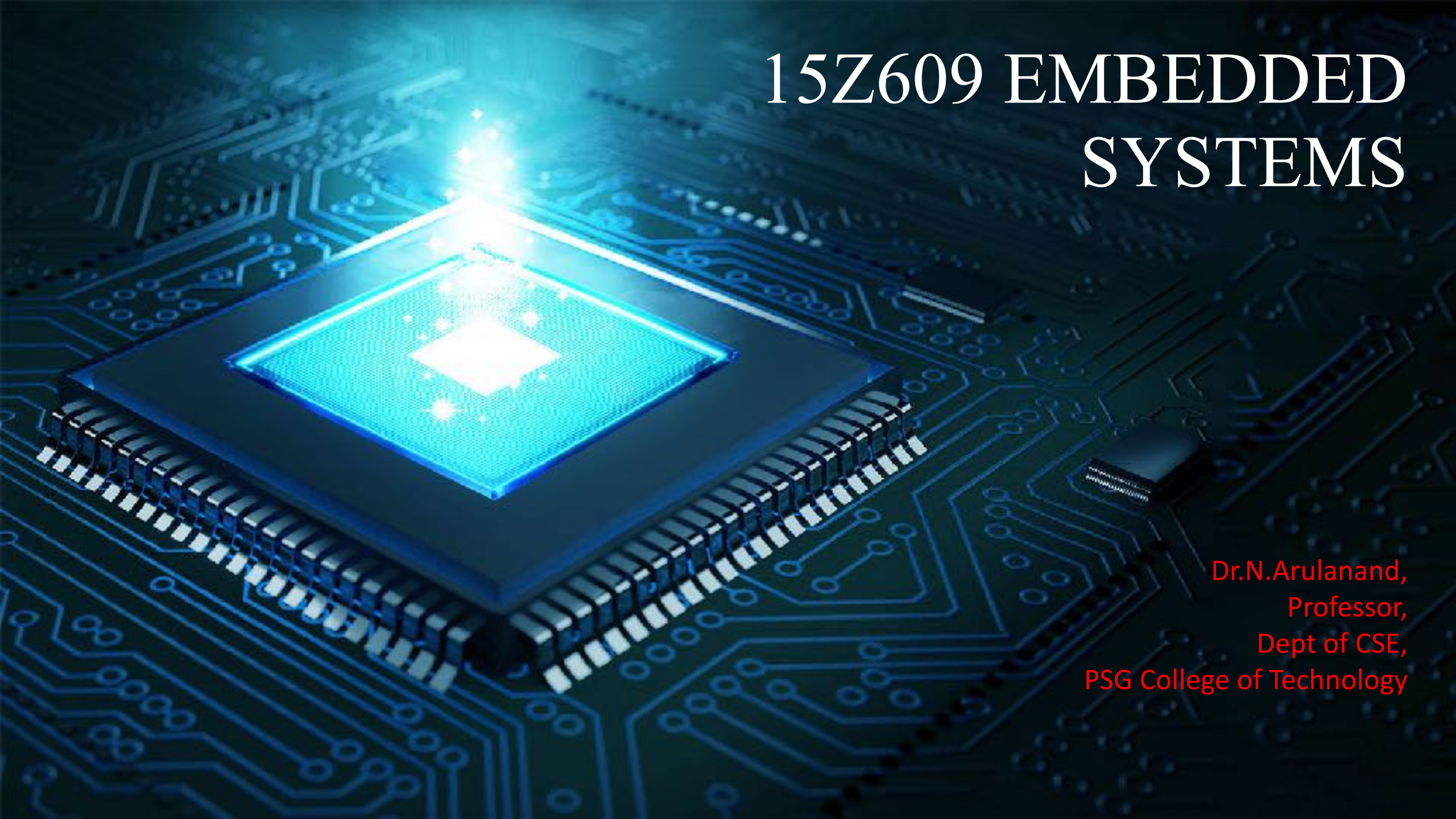
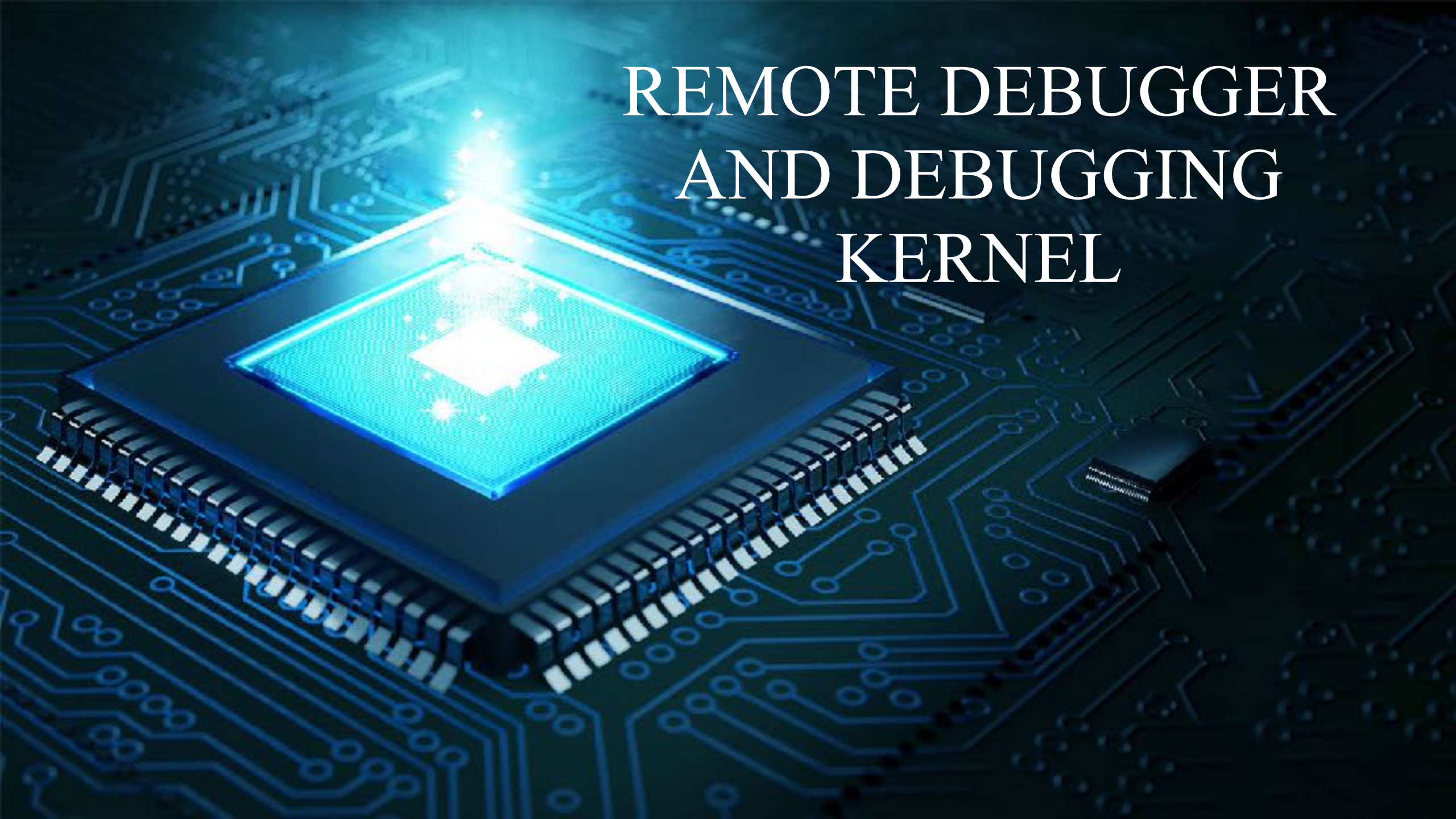


15Z609 EMBEDDED SYSTEMS

A glowing blue microchip is the central focus, resting on a dark blue circuit board with intricate white traces. The chip itself is a square with a grid of pins around its perimeter. A bright blue light emanates from the center of the chip, creating a lens flare effect. The background is a deep blue, and the overall aesthetic is high-tech and futuristic.

Dr.N.Arulanand,
Professor,
Dept of CSE,
PSG College of Technology

REMOTE DEBUGGER AND DEBUGGING KERNEL



DEBUGGING THE EMBEDDED SOFTWARE

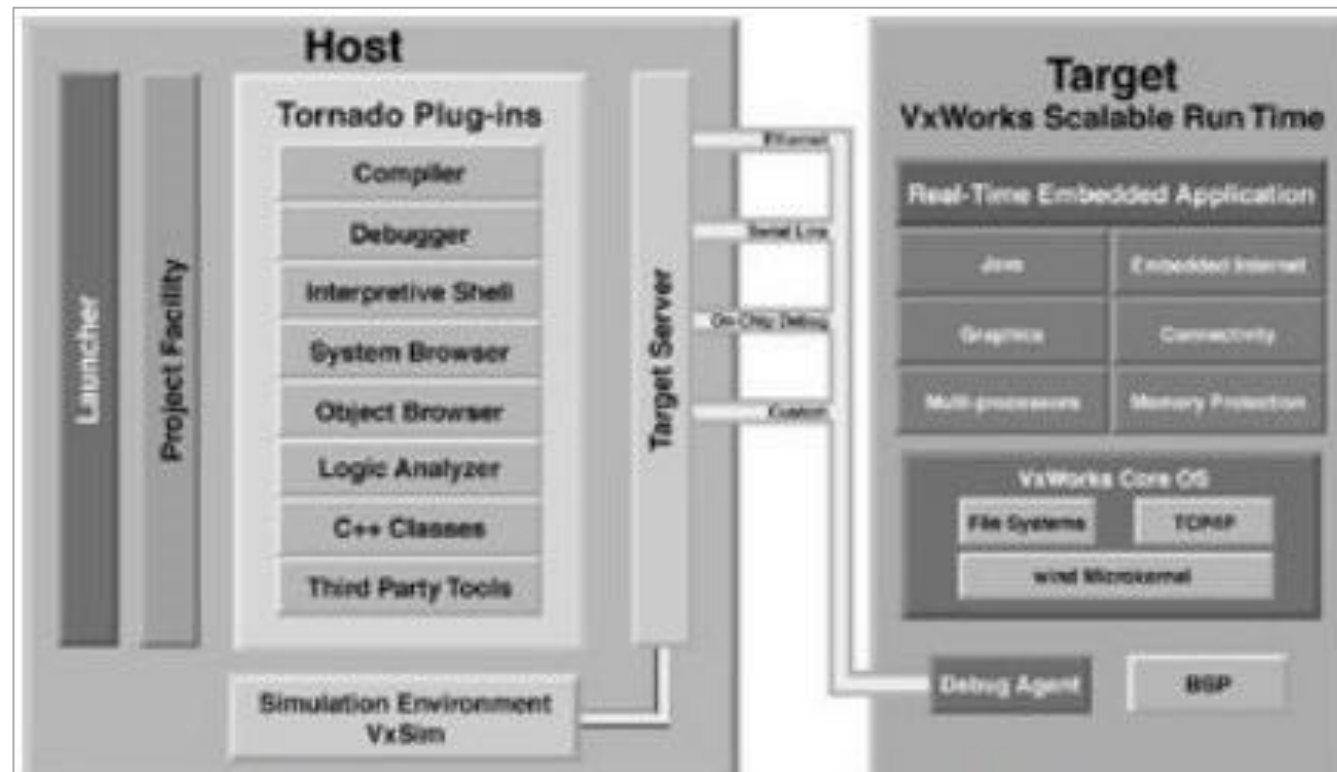
- **Debugging** - Process of eliminating bugs or errors in the software
- The software written to run on embedded systems may contain errors and hence needs debugging.
- **Difficulty** – In case of embedded systems, the difficulty is to find out the error or bug itself.
- Once the embedded systems starts functioning, there is no way for the user or programmer to know the **internal state** of the components of the target board.

REMOTE DEBUGGER

- Embedded platforms are too **resource-limited** and specialized to support a full-featured debugger.
- Debuggers for embedded systems address this limitation by **distributing** the debugger.
- A portion of the debugger resides on the host computer.
- The other portion resides on the target system.
- The portion in the host computer is called **Debugger Front End or GUI**.
- The portion in the target system is called **Target agent or Debug Kernel**.

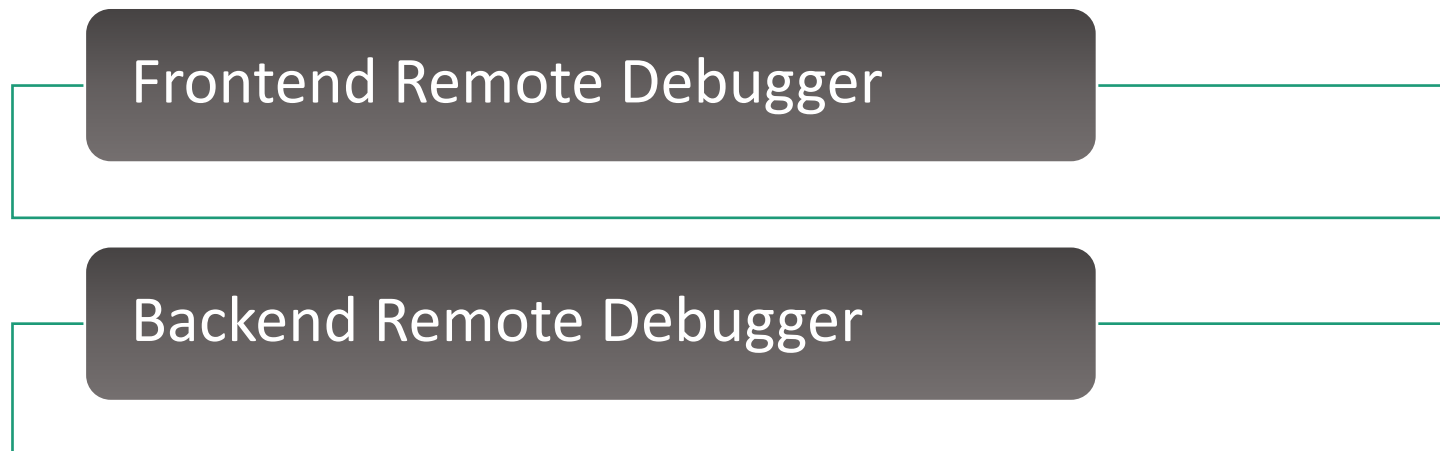
REMOTE DEBUGGER

- The two elements of the debugger communicate with each other over a communication channel such as a **Serial port** or **Ethernet port**.



SOFTWARE INTERFACE

- The software interface of the remote debugger has **GUI-based main window** and several smaller windows for the source code, register contents and other information about the executing program.
- It contains two pieces of software.



FRONT END & BACK END REMOTE DEBUGGER

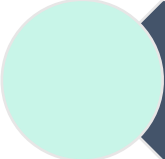

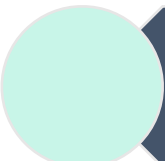


Frontend Remote Debugger:

- Runs on the host computer

Backend Remote Debugger:

- Runs on the target processor.
- Provides a low-level control of the target processor which is called the **debug monitor**.
- Debug monitor is a piece of software that has been designed specifically for use as a debugging tool for processors and chips.
- It is **automatically started** when the processor is reset.
- It monitors the communication link to the host computer and responds to requests from the remote debugger running there.

RUN CONTROL SERVICES

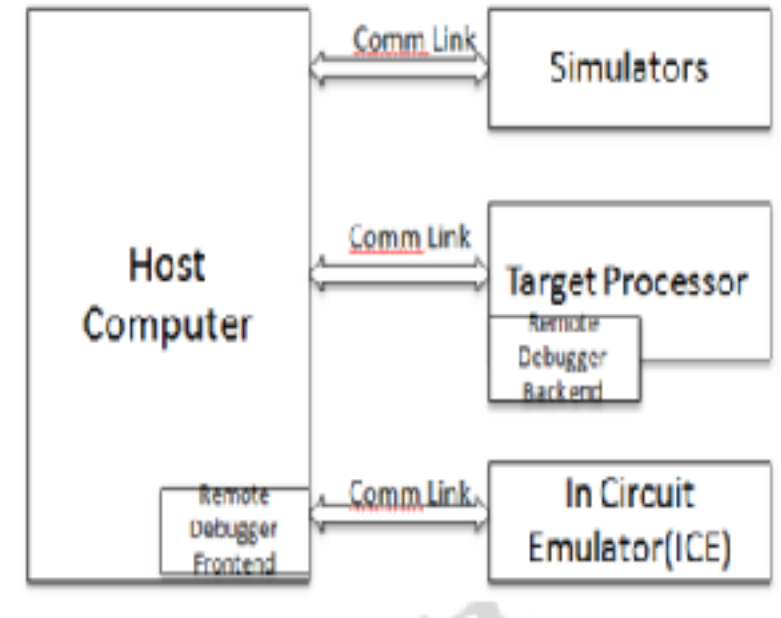
-  Setting Breakpoints
-  Loading programs from the host
-  Viewing or modifying memory & registers
-  Running from an address
-  Single-stepping the processor

HOST BASED DEBUGGING

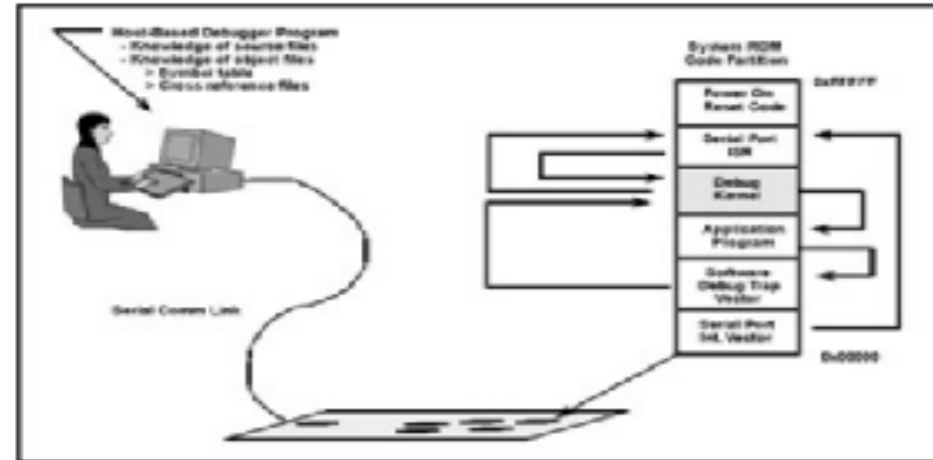
- A Debugger or debugging tool is a computer program that is used to test and debug other programs.
- A Remote Debugger contains a hardware interface between the host computer and the target embedded system.
- The greatest source of problems for host-based debugging derives from two architectural characteristics: word size and byte order.
- Another possible solution is for the software team to use Instruction Set Simulators (ISS) to allow them to compile their target code for their chosen microprocessor but execute the code on their workstations.
- The ISS is a program that creates a virtual version of the microprocessor

DEBUG KERNELS

- Debugging an embedded system is similar to debugging a host based application. Many embedded systems are not possible to debug unless they are operating at full speed. Hence debugging of an embedded system uses host computer
- The debugger can exist as two pieces, a debug kernel in the target and a host application that communicates with it and manages the source database and symbol tables.
- Debugger and software being debugged are executing on two different computer systems. It supports higher level of interaction between host and target.
- It allows -Start/restart/kill, and stepping through program. - Software breakpoints. - Reading/writing registers or data at specified address.



- The debug kernel requires two resources from the target.
 - Interrupt vector
 - Software interrupt



- The interrupt vector for the serial port forces the processor into the serial port ISR, which also becomes the entry point into the debugger
- 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
- The debug kernel is similar to an ISR in many ways. An interrupt is received from a device, such as the serial port, which happens to be connected to the designer's host computer.
- Just like an ISR, the arrival of a command from the host computer stops the execution of the application code and can cause the processor to enter the debug kernel ISR. The machine context is saved, and the debugger is now in control of the target.

BREAKPOINTS

- A breakpoint is a location where an action occurs, at which point the program stops executing. Although a trace's behavior is different from that of a breakpoint, traces and breakpoints share similar event handlers.
- The first breakpoint type that is used and generally preferred is a hardware breakpoint. Every microcontroller has comparators which are part of the debugging module
- A software breakpoint is typically an instruction that temporarily replaces an instruction in RAM that is either an illegal instruction and causes a fault or is designed to cause the application to break.
- A perfect example is the BKPT instruction in the ARM instruction set. When the CPU reaches this instruction, it halts execution.

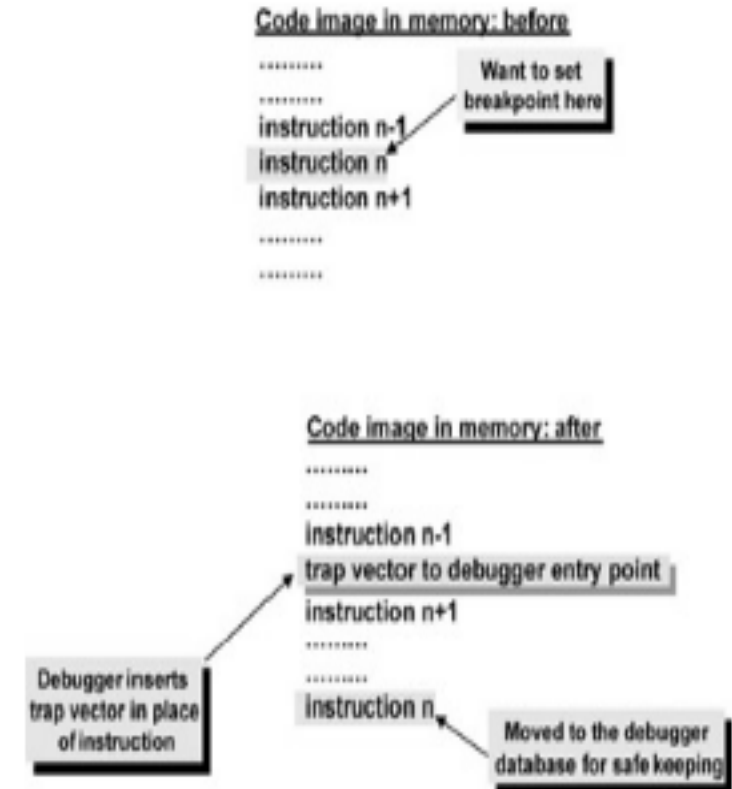


Figure 6.6: Breakpoints.
How a debugger sets a breakpoint in read/write memory.

- Software breakpoints can only be used for application code that reside in RAM. The reason is that an instruction is literally swapped out for the breakpoint instruction. Once a developer steps past the BKPT, the originally code that would have executed at that location is ran.
- Another obvious problem with this mechanism is that you need to be able to replace the user's instruction code with the trap code, thus implying that you can read and write to this memory region. If the code you're trying to debug is in true ROM or EPROM, you can't get there from here. You'll need to use a RAM-based ROM emulation device to give you the ability to replace user code with breakpoint traps.
- Flash breakpoints allow a developer to create unlimited breakpoints for applications that are running from flash. Just like a regular software breakpoint, a flash breakpoint has the ability to have a nearly endless number of breakpoints. They also can work on a microcontrollers internal and external flash memory

Advantage 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

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 environment

Disadvantage of the debug kernel

Depends on a stable memory sub system in the target and is not suitable 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 single step or insert breakpoints

Requires that the target has additional services, which, for many target systems, is not possible to implement

Debugger might not always have control of the system and depends on code being “well behaved”

PROM Programmer

A **PROM Programmer** (Programmable Read-Only Memory Programmer) is a hardware device used to **write data into programmable memory chips**, such as **PROM, EPROM, EEPROM, and Flash memory**. It is essential for embedded systems, firmware development, and hardware testing.

Example Use Case

◆ Burning Bootloader for Arduino or Raspberry Pi EEPROM

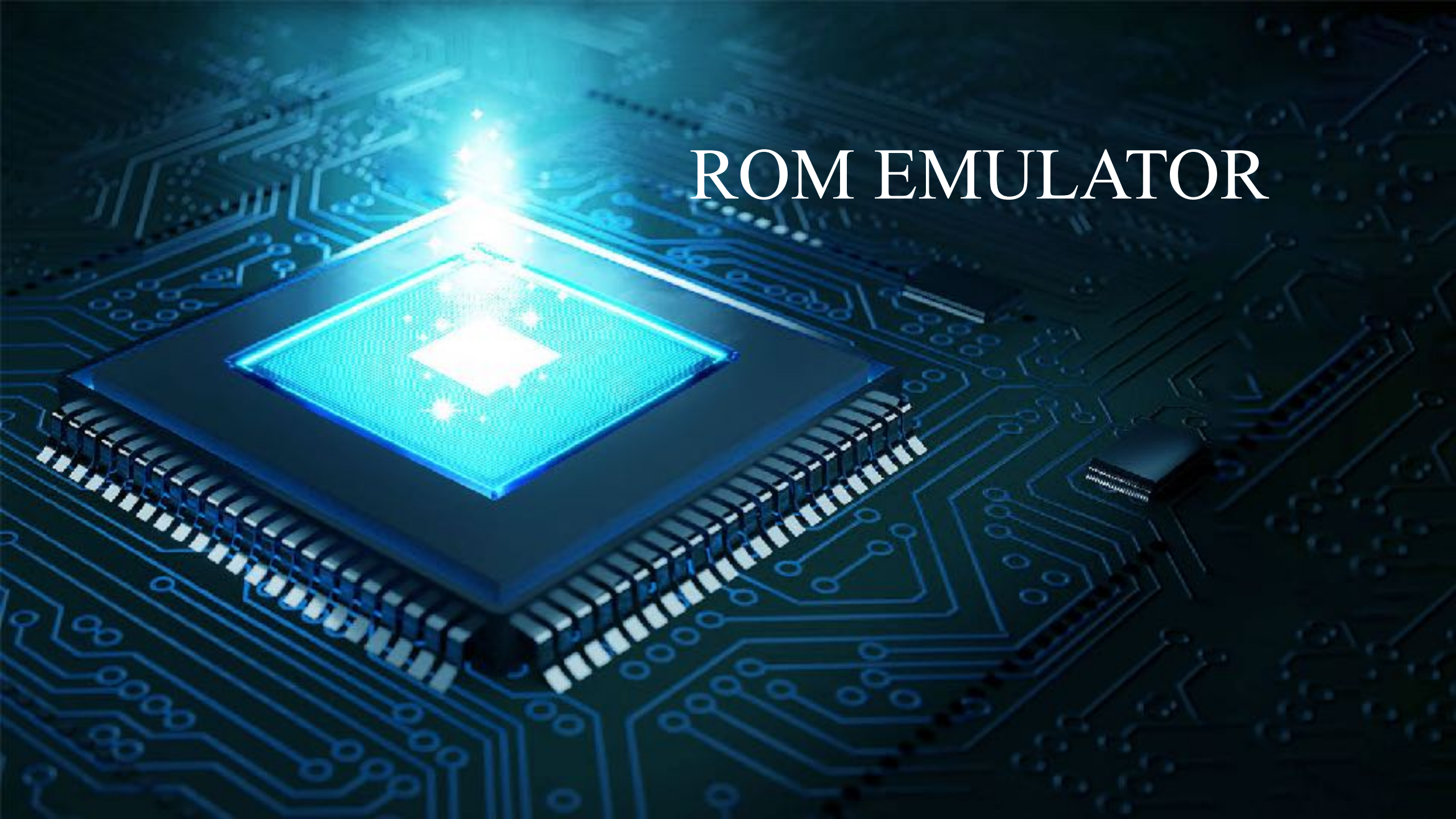
- 1 Write a bootloader onto an EEPROM using a PROM programmer.
- 2 Install the EEPROM in the microcontroller system.
- 3 The system boots from the programmed firmware.

● ◆ Restoring BIOS Firmware on a Motherboard

- 1 Use a PROM programmer to reflash the BIOS chip.
- 2 Replace the corrupted chip in the motherboard.
- 3 System boots with restored BIOS.



ROM EMULATOR



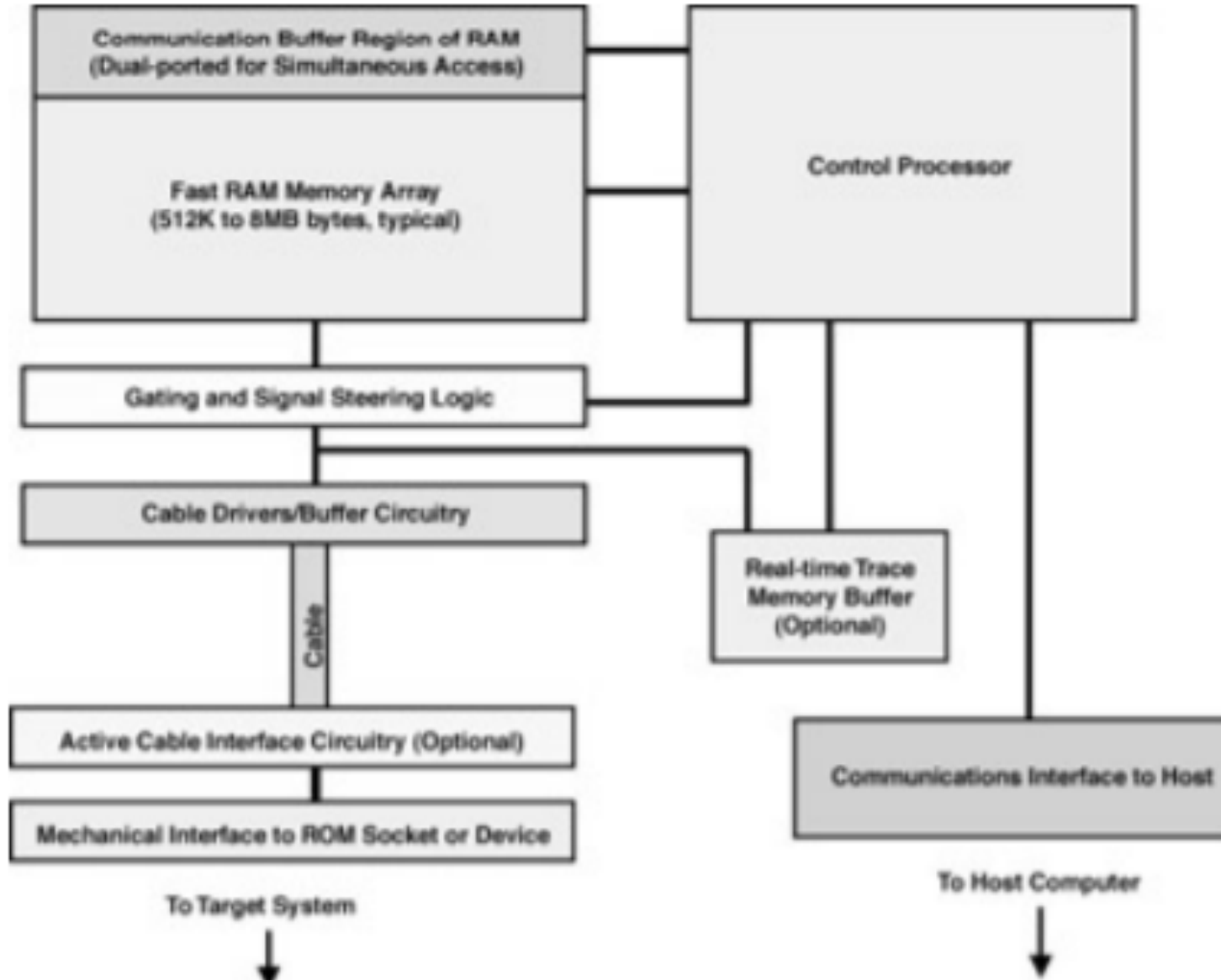
ROM EMULATOR

A ROM emulator is an electronic device used in embedded system development to simulate ROM.

The ROM emulator contains the following system elements:

- Cabling device
- Fast RAM
- Local control processor
- Communications port
- Trace memory and Flash Programming Algorithms

FUNCTIONAL BLOCK DIAGRAM



FUNCTIONS

- Breakpoints can be easily set in ROM emulator.
- If the debugger has been ported to work with the ROM emulator, the code substitution can be accomplished via the emulator control processor instead of by the target processor running in the debug kernel.
- It can be difficult to interface to the ROM emulator if the hardware designer didn't connect a write signal to the ROM socket. Most ROM emulators have a method of writing to ROM by executing a sequence of ROM read operations.
- Writing to ROM problem is solved.
- The ROM emulator can create a virtual UART port to the host computer.

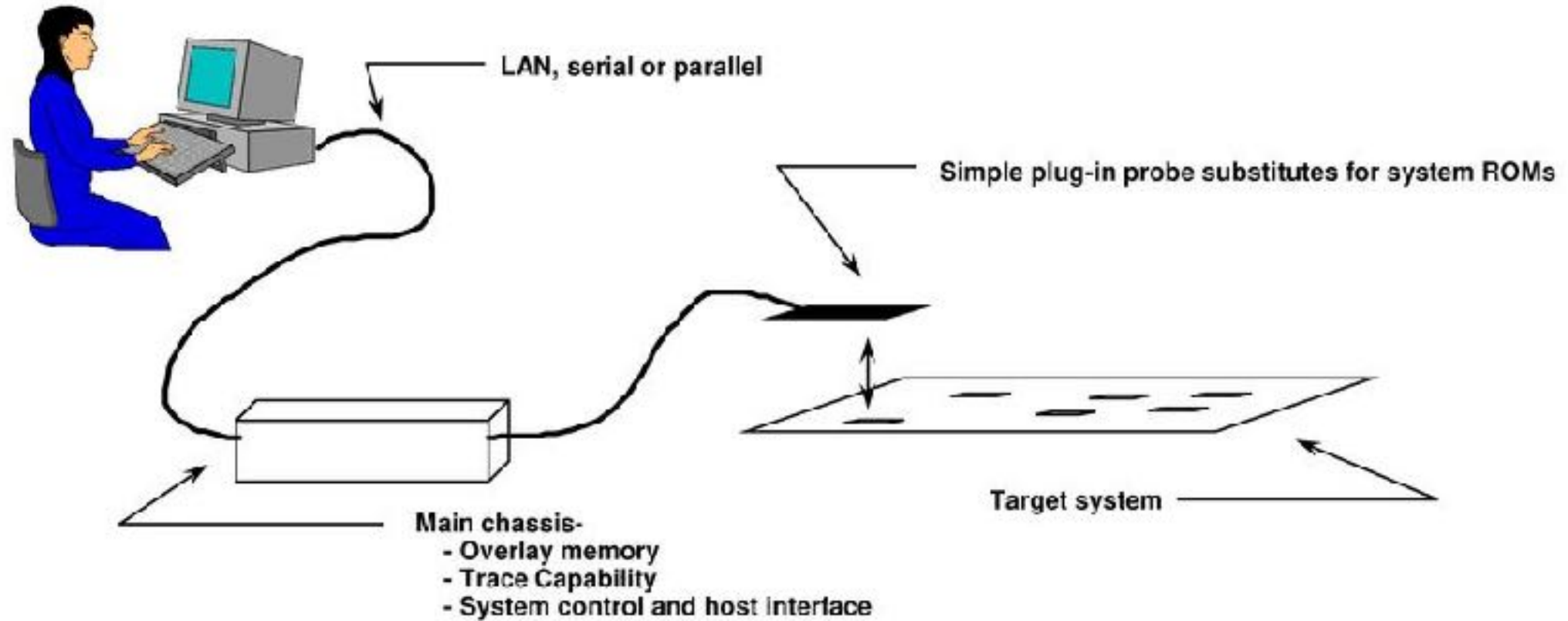
Advantages of ROM Emulator

Feature	Benefit
Real-Time Execution	No need to burn firmware repeatedly
In-Circuit Emulation	Works directly on target hardware
On-the-Fly Updates	Modify firmware without restarting
Breakpoint Support	Debug efficiently with debuggers
High-Speed Transfers	Faster than physical ROM flashing
USB & Network Access	Remote debugging support
Logging & Monitoring	Tracks memory access & errors
Multi-Architecture Support	Works with various MCUs

When to Use a ROM Emulator?

- When **debugging firmware** without wearing out Flash memory
- When testing different firmware versions **without reprogramming**
- When working with **legacy systems that use ROM chips**
- When **developing & testing bootloaders**

SCHEMATIC REPRESENTATION



Introduction to Embedded Systems

ADVANTAGES

- cost effective.
- Compatible with many different memory configurations.
- Can download large blocks of code to target system at high speed.
- Can trace ROM code activity in real time.
- Can be integrated with hardware and software tools like commercially available debuggers.

DISADVANTAGES

- Requires target system memory in a stable condition.
- Feasible only if embedded code is contained in standard ROMs.
- Real-time trace is possible only if program executes directly out of ROM
- Many targets transfer code to RAM for performance reasons