International Journal for Modern Trends in Science and Technology, 7(05): 225-232, 2021

ISSN: 2455-3778 online
DOI: https://doi.org/10.46501/IJMTST0705038
Available online at:http://www.ijmtst.com/vol7issue05.html

Efficiency of Embedded Software Development Tools and Debugging Techniques

[1]M Ram Gopal [2]Koneru Lakshmi Sowjanya

[1]Asst Professor Department of CSE, PVP Siddhartha Institute of Technology, Vijayawada, Andhra Pradesh, India
[2]Asst Professor Department of ECE, DIET, Vijayawada, Andhra Pradesh, India

To Cite this Article
M Ram Gopal and Koneru Lakshmi Sowjanya. Efficiency of Embedded Software Development Tools and Debugging Techniques. *International Journal for Modern Trends in Science and Technology* 7, 225-232 (2021).

Article Info
Received on 26-April-2021, Revised on 17-May-2021, Accepted on 26-May-2021, Published on 28-May-2021.

ABSTRACT

An Embedded System performs a single well defined task. The Embedded System can be described as one consisting of processor, associated peripherals and software used for a specific purpose. Embedded System is any device or collection of devices that contain one or more dedicated computer or microprocessor or microcontrollers. According Testing strategies debugging have to be accommodate testing. Testing is a discipline or methodology that encompasses set of principles to uncover the errors in programs. Debugging is the process of locate and fix an error in the program.

This paper presents about how the Embedded Software can be developed on host machine different from target machine on which the software will eventually be shipped to customer. Writing Embedded Software with few bugs is more important in embedded system development than in application because customers are intolerant of embedded system bugs and these bugs can be very hard to find. In this paper we will discuss about Embedded Software debugging tools and techniques.

Key words:*-Cross compiler, cross assembler, linker/ locator, Prom programmers, ROM emulators, In-circuit emulator and Monitors. Instruction set simulators, Laboratory tools*

1.Target System and Host computer: - The embedded system under development-may or may not have a keyboard, a screen, a disk drive, and other peripherals that are necessary for programming- may not have enough memory to run a programming editor- nobody has ever written an editor to run on the particular microprocessor in target system The standard platform used to develop the software and link to target system for debugging-programming work for embedded systems is done here-only after the program written, compiled, assembled and linked is it moved to target-It is also known as work station

Native tools:-Hosts are coming with compilers, assemblers, linkers and so on for building embedded programs that will run on the host. These tools are called the native tools.

Cross compilers: It is a compiler capable of creating executable code for a platform other than the one on which the compiler is run. A compiler that runs on one computer but produces object code for a different type of computer.It is a tool that one must use for a platform where it is inconvenient or impossible to compile on that platform, like microcontrollers that run with a minimal amount of memory for their own purpose.For example The native compiler on a Windows NT system based on an Intel Pentium i.e. builded programs are intended to run on an Intel Pentium. This compiler is useful if your target microprocessor is a Pentium, but it is useless

if your target microprocessor is something else says a Motorola, etc. A compiler runs (Pentium instructions) on your host system but produces binary instructions that will understood by your target microprocessor

Pitfalls:-In theory a program that compiles without error on your native compiler should also compile without error run on the cross compiler but in practice you should expect that certain constructions accepted by one compiler will not be accepted by another. Example In case of usage of functions without declaration, or if you declare with older styles2. The variables declared as int may be one size on the host and a different size on the target.3. Structures may be packed differently on the two machines.

2.Cross Assemblers and Tool chains:

Cross assembler An assembler that generates machine language (binary instructions) for a different type of computer than the one the assembler is running in. An assembler that generates machine language for a different type of computer than the one the assembler is running in. It is used to develop programs for Computers on a chip or microprocessors used in specialized applications that are either too small or are otherwise incapable of handling the development software.

3. Tool chain:-

A tool chain is the set of computer programs (tools) that are used to create a product (typically another computer program or system of programs). The tools may be used in a chain, so that the output of each tool becomes the input for the next, but the term is used widely to refer to any set of linked development tools. Differences between native linker and linker/locators

Native linker: - A native linker is a program that combines object modules to form an executable program. Many programming languages allow you to write different pieces of code, called modules, separately. This simplifies the programming task because you can break a large program into small, more manageable pieces. Eventually, though, you need to put all the modules together. This is the job of the linker. It is also known as link editor and binder. Loader will read the the file which is created by native linker whenever the user requests to run the program.. The loader finds memory into which to load the program and copies the program from the disk into the memory and then do various processing before starting the program.

Linker/locators:- The Job of a cross compiler is read the source file and produce an object file suitable for a linker. Linkers for an embedded system are often called locators or linker/locators or cross linker. In embedded system there is no loader. The locator creates a file that will be used by some program that copies the output to target program. Later the output from the locator will have to run on its own. The first differences between native linker and linker/locators are the nature of the output files they create.

The figure shows the process of building application software with native tools. The problem in that tool chain must solve is that many microprocessor instructions contain the addresses of their operands. For example, the Move instruction in ABB.C contains the register R1 holds the address of the variable id. Similarly the binary Call instruction that contains address of first. So here the problem is finding the address of variable or function. The process of solving this problem is called address resolution. When it is compiling ABB.C, the compiler has no idea what the addresses of id and first will be; therefore it leaves flags in ABB.OBJ for the linker. That is address of variable or function must be patched to instructions. When it is compiling AXX.C the flag corresponding to first indicating the location of first within the object file AXX.OBJ. When the linker put the two object files together, it figure out id and first are in relation to the start of the executable image and places that information in the executable file. After the loader copies the program into memory, it knows exactly where id and first are in memory and it can fix up the call and move instructions that came from ABB.C with those addresses.

Error!

Loader deal with CALL instructions in the finished program that call operating system functions. It is

possible for the loader to know where those operating system functions are and where the user defined variables and functions are located. In most embedded systems there is no loader. When the locator is done, its output will be copied onto target system. Therefore, the locator must know where in memory the program will reside and fix up all the addresses. Locators have mechanisms that allow you to tell them where the program will be on the target system. Locators use any number of different

output file formats and various tools which will accept various file formats. One common format is a file that is binary image which is copied onto ROM. Two other formats are Intel Hex File Format and Motorola S-Record Format.

PROM Programmer:-

The way to get the software from the locator output file into the target system is to use the file to create a ROM or PROM. PROM Programmer is a device that writes instructions and data into PROM chips. The bits in a new PROM are all 1s (continuous lines). The PROM programmer creates only 0s by "blowing" the middle out of the 1s. Some earlier units were capable of programming both PROMs and EPROM's. Short for programmable read-only memory, a memory chip on which data can be written only once. Once a program has been written onto a PROM, it remains there forever. Unlike RAM, PROMs retain their contents when the computer is turned off. The difference between a PROM and a ROM (read-only memory) is that a PROM is manufactured as blank memory, whereas a ROM is programmed during the manufacturing process. To write data onto a PROM chip, you need a special device called a PROM programmer or PROM burner. The process of programming a PROM is sometimes called burning the PROM. An EPROM (erasable programmable read-only memory) is a special type of PROM that can be erased by exposing it to ultraviolet light. Once it is erased, it can be reprogrammed. An EEPROM is similar to a PROM, but requires only electricity to be erased. A device that writes a program onto a PROM chip. If you want to use PROM Programmer for debugging purposes , it is useful to build versions of the target system in which the PROM is placed in a socket on the target system rather than being soldered directly into the circuit. Then when you find a bug, you can remove the PROM from the target system and put into the eraser (if it is EPROM) or into waste basket, program a new PROM with software that has the bug fixed and put that PROM into the socket.

ROM emulator:

A circuit that helps debugs a ROM chip by simulating the ROM with RAM. The RAM circuit plugs into the ROM socket. Since RAM can be written over, whereas ROM cannot, programming changes can be made easily. ROM emulator is a device that replaces the ROM in the target system. The emulator looks like ROM.

ROM emulator contains a large box of electronics and a serial port or a n/w connection through which it can be connected to host. The s/w running on your host can send files created by locator to the ROM emulator which will act as ROM that has been programmed with the s/w you have written. The most common use for a voltmeter is to determine whether or not the chips in your circuit have power. A system can suffer power failure for any number of reasons- broken leads, incorrect wiring, blown fuses, failure to plug-in or turn on the power supply, etc.-and no amount of s/w effort will make such a system work. Part of the problem of making an accurate voltmeter is that of calibration to check its accuracy. The way to use a voltmeter is to turn the power on, put one the meter probes on a pin that should be attached to VCC and the other on a pin that should be attached to ground.

4.Monitors:-

Monitor is a program that resides in the Target ROM and knows how to load new programs onto the system. A typical s/w allows you to send your s/w across a serial port and stores that s/w in the target RAM and then runs it. Sometimes monitor performs functions of a locator and offer few debugging techniques. You can write your own monitor program.

Laboratory Tools:-

An oscilloscope or scope is a graph- displaying device that graphs voltage versus time of an electrical signal.. An oscilloscope is an analog device which detects whether a signal is high or low but the signal's exact voltage.In most applications the graph shows how signals change over time: the vertical (Y) axis represents voltage and the horizontal (X) axis represents time. The intensity or brightness of the display is sometimes called the Z axis. (See Figure 1.) This simple graph can tell you many things about a signal. Here are a few: You can determine the time and voltage values of a signal.

- You can calculate the frequency of an oscillating signal.
- You can see the "moving parts" of a circuit represented by the signal
- You can tell if a malfunctioning component is distorting the signal.
- You can find out how much of a signal is direct current (DC) or alternating current (AC).

- You can tell how much of the signal is noise and whether the noise is changing with time.

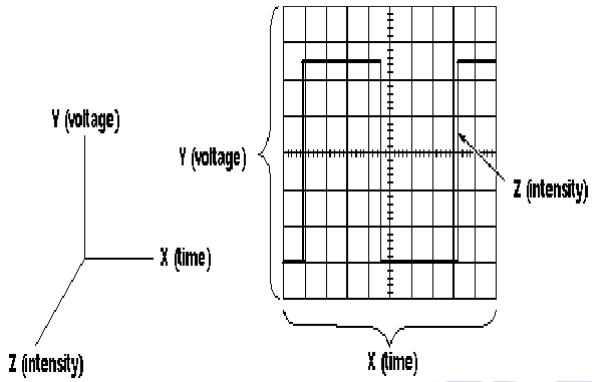


Figure1:X,Yand Z Components of a Displayed Waveform

An oscilloscope looks a lot like a small television set, except that it has a grid drawn on its screen and more controls than a television. The front panel of an oscilloscope normally has control sections divided into Vertical, Horizontal, and Trigger sections. There are also display controls and input connectors. See if you can locate these front panel sections in Figures 2 and 3 and on your oscilloscope.

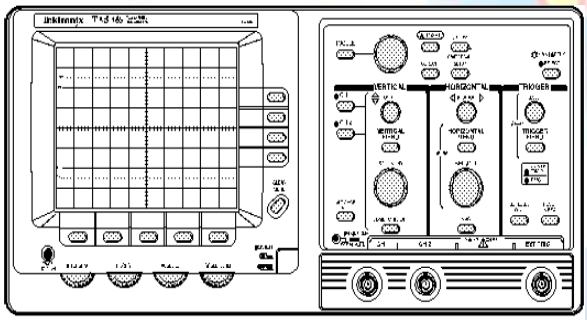
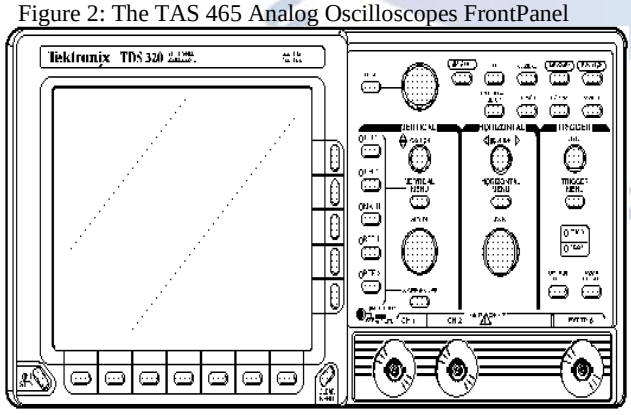


Figure 2: The TAS 465 Analog Oscilloscopes FrontPanel



Oscilloscopes are used by everyone from television repair technicians to physicists. They are indispensable for anyone designing or repairing electronic equipment.

The usefulness of an oscilloscope is not limited to the world of electronics. With the proper transducer, an oscilloscope can measure all kinds of phenomena. A transducer is a device that creates an electrical signal in response to physical stimuli, such as sound, mechanical stress, pressure, light, or heat. For example, a microphone is a transducer. An automotive engineer uses an oscilloscope to measure engine vibrations. A medical researcher uses an oscilloscope to measure brain waves. The possibilities are endless.

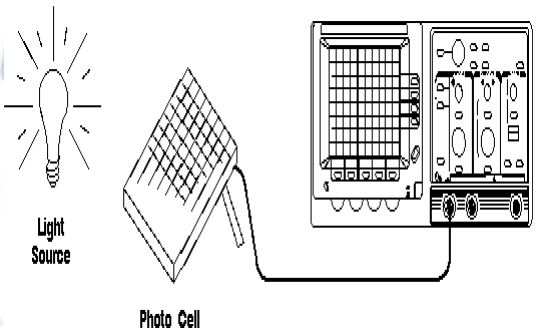


Figure 4: Scientific Data Gathered by an Oscilloscope

Setting Up the Oscilloscope

Properly grounding the oscilloscope protects you from a hazardous shock and grounding yourself protects your circuits from damage. If a high voltage contacts the case of an ungrounded oscilloscope, any part of the case, including knobs that appear insulated, it can give you a shock. However, with a properly grounded oscilloscope, the current travels through the grounding path to earth ground rather than through you to earth ground. To ground the oscilloscope means to connect it to an electrically neutral reference point (such as earth ground).Grounding is also necessary for taking accurate measurements with your oscilloscope. After plugging in the oscilloscope, take a look at the front panel. It is divided into three main sections labeled Vertical, Horizontal, and Trigger. Most oscilloscopes have at least two input channels and each channel can display a waveform on the screen. For each channel an input connectors is there on your oscilloscope. This is where you attach probes.

5.Display Controls:

Display systems vary between analog and digital oscilloscopes. Common controls

include:An intensity control to adjust the brightness of the waveform. As you increase the

sweep speed of an analog oscilloscope, you need to increase the intensity level.

A focus control to adjust the sharpness of the waveform. Digital oscilloscopes may not have a focus control. A trace rotation control to align the waveform trace with the screen's horizontal axis. The position of your oscilloscope in the earth's magnetic field affects waveform alignment. Digital oscilloscopes may not have a trace rotation control. Other display controls may let you adjust the intensity of the graticule lights and turn on or off any on-screen information (such as menus).

Vertical Controls

Use the vertical controls to position and scale the waveform vertically. Your oscilloscope also has controls for setting the input coupling and other signal conditioning,

Position and Volts per Division

The vertical position control lets you move the waveform up or down to exactly where you want it on the screen.The volts per division (usually written volts/div) setting varies the size of the waveform on the screen. A good general purpose oscilloscope can accurately display signal levels from about 4 millivolts to 40 volts. Often the volts/div scale has either a variable gain or a fine gain control for scaling a displayed signal to a certain number of divisions. Use this control to take rise time measurements.

Horizontal Controls

Use the horizontal controls to position and scale the waveform horizontally. Figure 5 shows a typical front panel and on-screen menus for the horizontal controls.

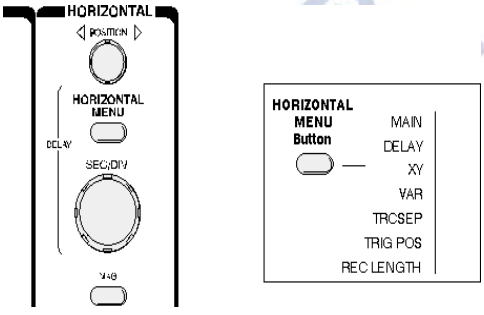


Figure 5: Horizontal Controls

6 Position and Seconds per Division

The horizontal position control moves the waveform from left and right to exactly where you want it on the screen.

The seconds per division (usually written as sec/div) setting lets you select the rate at which the waveform is drawn across the screen (also known as the time base setting or sweep speed). This setting is a scale factor. For example, if the setting is 1 ms, each horizontal division represents 1 ms and the total screen width represents 10 ms (ten divisions). Changing the sec/div setting lets you look at longer or shorter time intervals of the input signal.

As with the vertical volts/div scale, the horizontal sec/div scale may have variable timing, allowing you to set the horizontal time scale in between the discrete settings. The trigger position control may be located in the horizontal control section of your oscilloscope. It actually represents "the horizontal position of the trigger in the waveform record." Horizontal trigger position control is only available on digital oscilloscopes. Varying the horizontal trigger position allows you to capture what a signal did before a trigger event (called pretrigger viewing). the trigger merely tells the oscilloscope to save the present data in memory.

Trigger Controls

The trigger controls let you stabilize repeating waveforms and capture single-shot waveforms. The trigger makes repeating waveforms appear static on the oscilloscope display. For edge triggering, the trigger level and slope controls provide the basic trigger point definition. The trigger circuit acts as a comparator. You select the slope and voltage level of one side of the comparator. When the trigger signal matches your settings, the oscilloscope generates a trigger. The slope control determines whether the trigger point is on the rising or the falling edge of a signal. A rising edge is a positive slope and a falling edge is a negative slope. The level control determines where on the edge the trigger point occurs.

Trigger Sources

The oscilloscope does not necessarily have to trigger on the signal being measured. Several sources can

trigger the sweep:

Any input channel

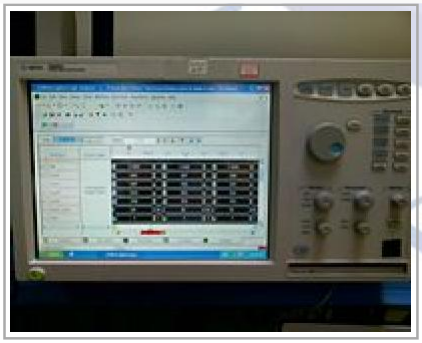
An external source, other than the signal applied to an input channel

The power source signal A signal internally generated by the oscilloscope Most of the time you can leave the oscilloscope set to trigger on the channel displayed. Note that the oscilloscope can

use an alternate trigger source whether displayed or not. So you have to be careful not to unwittingly trigger on, for example, channel 1 while displaying channel

Trigger Modes 2.

The trigger mode determines whether or not the oscilloscope draws a waveform if it does not detect a trigger. Common trigger modes include normal and auto. In normal mode the oscilloscope only sweeps if the input signal reaches the set trigger point; otherwise (on an analog oscilloscope) the screen is blank or (on a digital oscilloscope) frozen on the last acquired waveform. Normal mode can be disorienting since you may not see the signal at first if the level control is not adjusted correctly. Auto mode causes the oscilloscope to sweep, even without a trigger. If no signal is present, a timer in the oscilloscope triggers the sweep. This ensures that the display will not disappear if the signal drops to small voltages. It is also the best mode to use if you are looking at many signals and do not want to bother setting the trigger each time. logic analyzer is an electronic instrument that displays signals in a digital circuit that are too fast to be observed and presents it to a user so that the user can more easily check correct operation of the digital system. They are typically used for capturing data in systems that have too many channels to be examined with an oscilloscope. Software running on the logic analyzer can convert the captured data into timing diagrams, protocol decodes, state machine traces, assembly language, or correlate assembly with source- level software.



LogicAnalyzer

integrated development environments (IDEs).Also called link editor and binder, a linker is a program that combines object modules to form an executable program. Many programming languages allow you to write different pieces of code, called modules, separately. This simplifies the programming task because you can break a large program into small, more manageable pieces. Eventually, though, you need to put all the modules together. This is the job of the linker. A logic analyzer can trigger on a complicated sequence of digital events, and then capture a large amount of digital data from the system under test (SUT). The best logic analyzers behave like software debuggers by showing the flow of the computer program and decoding protocols to show messages and violations.

Uses:Many digital designs, including those of ICs, are simulated to detect defects before the unit is constructed. The simulation usually provides logic analysis displays. Often, complex discrete logic is verified by simulating inputs and testing outputs using boundary scan. Logic analyzers can uncover hardware defects that are not found in simulation. These problems are typically too difficult to model in simulation, or too time consuming to simulate and often cross multiple clock domains. Field have become a common measurement point for logic analyzers.

Probes

Today, the most common method of data capture for logic analyzers is through a probe. A logic analyzer can measure anything electronic if it has the proper probe connected. Mostly logic analyzer measure data buses. The probes try to tap into the electronic signals being passed Integrated development environments (IDEs) combine the features of many tools into one complete package. They are usually simpler and make it easier to do simple tasks, such as searching for content only in files in a particular project. Tools were originally simple and light weight. As some tools have been maintained, they have been integrated into more powerfulAn Instruction Set Simulator (ISS) is a simulation model, usually, but by no means always, coded in a high-level language, which mimics the behavior of a mainframe or microprocessor by "reading" instructions and maintaining internal variables which represent the processor's registers.

8.Implementation

An ISS is often provided with a debugger in order for a software engineer to debug the program prior to obtaining target hardware. GDB is one of debuggers which have compiled-in ISS. It is sometimes integrated with simulated peripheral circuits such as timers, interrupts, serial port, general I/O port,

etc to mimic the behavior of microcontroller. The basic instruction simulation technique is the same regardless of purpose - first execute the monitoring program passing the name of the target program as an additional input parameter.

The target program is then loaded into memory, but control is never passed to the code. Instead, the entry point within the loaded program is calculated and a pseudo Program status word (PSW) is set to this location. A set of pseudo registers are set to what they would have contained if the program had been given

control directly.

Thereafter, execution proceeds as follows:

1. Determine length of instruction at pseudo PSW location (initially the first instruction in the target program). If this instruction offset within program matches a set of previously given "pause" points, set "Pause" reason, go to 7.

2. Fetch" the instruction from its original location (if necessary) into the monitor's memory. If trace is available and on store program name, instruction offset and any other values.

3. depending upon instruction type perform pre-execution checks and execute. If the instruction cannot proceed for any reason (invalid instruction, incorrect mode etc) go to 7. If the instruction is about to alter memory, check memory destination exists (for this thread) and is sufficiently large. If OK, load appropriate pseudo registers into temporary real registers, perform equivalent move with the real registers, save address and length of altered storage if trace is "on" and go to 4. If the instruction is a "register-to-register" operation, load pseudo registers into monitors real registers, perform operation, store back to respective pseudo registers, go to 4. If the instruction is a conditional branch, determine if the condition is satisfied: if not go to 4, if condition IS satisfied, calculate branch to address, determine if valid (if not, set error = "Wild branch") and go to 7. If OK, go to 5. If instruction is an operating system call, do real call from monitoring program by "faking" addresses to return control to monitor program and then reset pseudo registers to reflect call; go to 4.

4. Add instruction length to current Pseudo PSW value.

5. Store next address in Pseudo PSW.

6. Go to 1.

7. Halt execution.

For test and debugging purposes, the monitoring program can provide facilities to view and alter registers, memory, and re-start location or obtain a mini core dump or print symbolic program names with current data values. It could permit new conditional "pause" locations; remove unwanted pauses and such like. Instruction simulation provides the opportunity to detect errors BEFORE execution which means that the conditions are still exactly as they were and not destroyed by the error. A very good example from the IBM S/360 world is the following instruction sequence that can cause difficulties debugging without an instruction simulation monitor.

9.Overhead

The number of instructions to perform the above basic "loop" (Fetch/Execute/calculate new address) depends on hardware but it could be accomplished on IBM S/360/370/390/ES9000 range of machines in around 12 or 13 instructions for many instruction types. Checking for valid memory locations or for conditional "pauses add considerably to the overhead but optimization techniques can reduce this to acceptable levels. For testing purposes this is normally quite acceptable as powerful debugging capabilities are provided including instruction step, trace and deliberate jump to test error routine (when no actual error). In addition, a full instruction trace can be used to test actual (executed) code coverage.

Added benefits

Occasionally, monitoring the execution of a target program can help to highlight random errors that appear (or sometimes disappear) while monitoring but not in real execution. This can happen when the target program is loaded at a different location than normal because of the physical presence of the monitoring program in the same address space. If the target program picks up the value from a "random" location in memory (one it doesn't 'own' usually), it may for example be nulls (X"00") in almost every normal situation and the program works OK. If the monitoring program shifts the load point, it may pick up say X"FF" and the logic would cause different results during a comparison operation. Alternatively, if the monitoring program is now occupying the space where the value is being "picked up" from, similar results might

occur. Re-entrancy bugs: accidental use of static variables instead of "dynamic" thread memory can cause re- entrancy problems in many situations. Use of a monitoring program can detect these even without a storage

protect key

An in-circuit emulator (ICE) is a hardware device used to debug the software of an embedded system. It is usually in the form of bond-out processor which has many internal signals brought out for the purpose of debugging. These signals tell about the state of a processor. Embedded systems present special problems for a programmer, because they usually lack keyboards, screens, disk-drives and other helpful user interfaces and storage devices that are present on business computers.

In-circuit emulation can also refer to the use of hardware emulation, when the emulator is plugged into a system (not always embedded) in place of a yet-to-be-built chip (not always a processor). These in-circuit emulators provide a way to run the system with "live" data while still allowing relatively good debugging capabilities. It can be useful to compare this with an in-target probe (ITP) sometimes used on

enterprise servers.

Advantages

Virtually all embedded systems have a hardware element and a software element, which are separate but tightly interdependent. The ICE allows the software element to be run and tested on the actual hardware on which it is to run, but still allows programmer conveniences to help isolate faulty code, such as "source-level debugging" Most ICEs consist of an adaptor unit that sits between the ICE host computer and the system to be tested. A header and cable assembly connects the adaptor to a socket where the actual CPU or microcontroller mounts within the embedded system. Recent ICEs enable a programmer to access the on- chip debug circuit that is integrated into the CPU via JTAG or BDM (Background Debug Mode) in order to debug the software of an embedded system. These systems often use a standard version of the CPU chip, and can simply attach to a debug port on a production system. They are sometimes called In-circuit Debuggers or ICDs, to distinguish the fact that they do not replicate the functionality of the CPU, but instead control an already existing, standard CPU. Since the CPU does not have to be replaced, they can operate on production units where the CPU is soldered in and cannot be replaced. An example is Microchip Technology's ICD, which interfaces with most recent PIC microcontrollers to debug software by attaching to the PIC's In-circuit programming/debugging port. The ICE emulates the CPU. From the system's point of view, it has a real processor fitted, but from the programmer's point of view the system under test is under full control, allowing the developer to load, debug and test code directly. Most host systems are ordinary commercial computers unrelated to the CPU used for development - for example, a Windows PC might be used to develop software for a system using a Free scale 68HC11 chip, which itself could not run Windows. The programmer usually edits and compiles the embedded system's code on the host system, as well. The host system will have special compilers that produce executable code for the embedded system. These are called cross compilers.

REFERENCES

[1] "An Embedded Software Primer", David E Simon, Pearson Education

[2] "Programming for Embedded Systems", DreamTech Software Team, Wiley Publications Inc.

[3] "Embedded Systems Building Blocks", Labrosse, CMP Publications

[4] "Embedded Systems", Rajkamal, Tata McGraw Hill

[5] "Embedded System Design", Frank Vahid, John Wiley Publications