

A PROCESSOR SOFTWARE DEVELOPMENT AND TEST APPARATUS

Field of the Invention

The present invention relates to apparatus used in the development and testing of firmware for embedded processor systems.

Description of the Prior Art

The use of embedded processor systems is widespread. Their applications range from simple consumer electronic appliances to advanced aeronautical systems. Apart from the hardware design, firmware programs have to be written in order for embedded processors to function according to the specifications.

The testing and debugging of firmware in real-time systems can be both time consuming and tedious. The processor employs time sharing strategies to service the various functional software modules of the system in real-time. Thus, it is often difficult to determine the current state of execution of the processor at any particular point in time.

At present, in-circuit-emulators are used to aid in the development and debugging of firmware programs with the help of code tracing and stepping facilities. In-circuit-emulators use remote pods that are plugged into the processor's socket to probe and examine the processor's signals. An interface allows the user to monitor and intervene in the operation of the processor. The in-circuit-emulator permits break points to be set at specific locations in the code. The contents of the internal memory and registers can be viewed only when the processor is in the halt state.

In-circuit-emulators function exclusively for the family of processors that they are designed to emulate. It is not possible to carry over the use of an emulator from one class of processors to another totally different set of processors. Emulators are therefore processor specific.

The software equivalent of the in-circuit-emulator can also be used to verify the functionality of firmware. A software emulator is a piece of software that is used to simulate the resources and behaviour of a specific processor. However, this method of

A - 1031000

testing is not capable of investigating the effects of the firmware on the associated hardware that is connected to the processor.

A hardware device that monitors the input and output signals of the target processor is used. This device allows data or command signals to be driven into the input pins of the processor, so that the response of the processor to a given input can be monitored.

The above mentioned hardware device, and an in-circuit-emulator are two independent firmware debugging systems. They can however, be used together to form an integrated development and test system.

The testing and debugging of firmware in embedded real-time systems is often tedious and time consuming. One such example is given in the field of consumer electronics.

Many electronic appliances contain a real-time clock with events that are triggered at specific clock values preset by the user. Testing and debugging carried out on such systems is time consuming because of the need to wait for real-time to pass before the event schedule can be triggered. When a change is made to the firmware, the system needs to be tested again. These lengthy repetitive test cycles significantly prolong the time needed to test the system. No provision is made for unsupervised verification of events related to the real-time clock.

One such example is the testing of the timer and clock sub-systems in a consumer audio product. The timer has wake-up and sleep modes. Specific times can be preset for the audio equipment to go into stand-by mode (sleep mode) or to turn on. To test such a system, a wake-up time or a sleep time is first set. Real-time needs to elapse before inspection can be made to see if the correct event occurs. The time to trigger the sleep or stand-by mode may be many hours away. This results in a long wait for time to elapse before the appropriate event happens. For example, if the current time is 0700 hours, and a wake-up time is set to be at 1900 hours, it would result in a 12 hour wait. This long wait is time consuming and needs to be reduced.

It is necessary to examine the contents of memory and registers during the debugging of a real-time system. Current in-circuit-emulators only allow the memory contents to be changed when the processor is in a halt or reset state. No modification of the memory by the in-circuit-emulator is possible when the processor is executing.

In-circuit-emulator equipment is often costly, and processor specific. In an environment whereby a wide assortment of processors are available for use, acquiring each corresponding emulator can be financially prohibitive. Time is also required to allow the user to gain familiarity with the operation of each new emulator. No emulator capable of targeting different families of processors is available. When a transition is made from the use of one processor to another in a typical system, the program variable assignments and label names employed in the current and previous set of firmware are usually different, as they are often written by different authors. A third party attempting to troubleshoot the firmware will experience difficulty in comprehending the code as the variables and labels used for the same functionality are named differently.

Accordingly, a low cost integrated apparatus for testing, debugging and monitoring the system under development, which includes a subset or full set of the functions of a generic emulator is a tool that can greatly enhance and expedite the development of firmware for embedded processor systems.

Summary of the Invention

In accordance with one aspect of the present invention there is disclosed a real-time monitoring apparatus for testing and debugging of firmware in an embedded processor based system, said apparatus comprising:

- a host computer sub-system;
- a target sub-system representing the embedded processor system;
- a communication link sub-system interconnecting the host computer sub-system and the target sub-system; and;

a monitoring sub-system residing on the host computer sub-system for monitoring operation of said target sub-system in real-time and retaining monitored results thereof.

In accordance with another aspect of the present invention there is disclosed a method for the standardised development of communication primitives to allow one of a plurality of target sub-systems, each having a different controlling processor, to be tested using a host monitoring system, said method comprising the steps of:

- (a) standardising variable names within each field of application of said target sub-systems and retaining same within said host monitoring system; and
- (b) standardising communication primitives for and amongst each of said different controlling processors and a controlling host processor of said host monitoring system and retaining same within each one of said target sub-systems.

In accordance with another aspect of the present invention there is disclosed a universal in-circuit-emulator apparatus for development and testing of one of a plurality of target systems, each of said target systems having a controlling target processor, said apparatus comprising:

- a user interface sub-system;
- a processor resource data depository sub-system;
- a test database engine sub-system;
- an auto tester sub-system;
- a generic emulator software engine sub-system;
- a generic emulator hardware sub-system; and
- a generic remote processor pod sub-system.

Arising from one embodiment the invention, an economical and efficient system capable of accessing and modifying memory and register contents in real-time has been devised, which is able to monitor the operation with any type of embedded processor.

The system includes a means to monitor the memory and register resources of the target processor in real-time, facilitating the verification of processor operations,

and a means by which different families of target processors can be monitored using the same host and communication sub-system.

The host processor sub-system includes a means to store and document test cases for specific tests, a means to carry out testing over extended periods of time without any human intervention, a means to describe the internal resources of the target processor to the monitoring system, a means to accept user input and provide feedback to the user, a means to encode user inputs, a means to send information to target sub-system.

The target processor sub-system includes a means to decode the information sent by the host monitoring sub-system, a means to write and read from memory and registers as requested by the monitoring sub-system, and a means to send information back to the monitoring sub-system.

To address the problem of having to acquire a new emulator whenever a new processor is used, a universal emulator apparatus is provided, which provides a means to debug target processors across different platforms and manufacturers by using only one set of hardware, a means to store and edit the internal resource configuration of the various families of processors in a processor data depository, a means to extract the relevant processor data from the depository to initialise and setup the universal emulator for use with a new target processor, a means to provide a standardised user interface regardless of the target processor being used, a means to read and write memory areas in real-time, while the program is executing and a means for the remote processor pod to accommodate processors with different packaging and pin configurations.

To address the issue concerning the varied use of variable assignments and label names by different developers across different processors, a means is used to standardise the name of variables and labels is provided in preferred embodiments.

Brief Description of Drawings

Fig. 1 is a schematic block diagram showing the various functional modules within the host and the target systems and how they interact with each other, according to an embodiment of the invention;

Fig. 2 is a schematic block diagram illustrating the shared use of the communication and control lines between the host, target and the external device, in a particular system;

Fig. 3 is a schematic block diagram depicting the interaction between the various functional modules of the universal emulator apparatus of the preferred embodiment; and

Fig. 4 is a schematic flowchart of a real-time operating system of the target processor.

Detailed Description of the Best Mode of Performing the Invention

Fig. 1 shows a host monitor system 10 interconnected to a target system 30 via communication protocol implemented using communication line 39. The host monitor system 10 is used for the software development and testing of the target system 30.

The host monitor system 10 consists of:

- a user command interface module 18;
- a communication module including a hardware interface 13 to the communication line 39, and a communication primitives module 14;
- a monitoring module 15;
- a database engine module 17;
- an auto test module 16;
- a set of target processor specifications 19;
- a host processor 11;
- a host bus 12 interconnecting the modules with the host processor 11.

The target system 30 of Fig. 1 consists of:

- a target processor 34, generally including RAM;
- processor firmware 35 for the target processor 34, generally formed in ROM internal or external to the target processor 34;

- system hardware 36 including all devices controlled by the target processor;
- a communication module including primitives 33 and a hardware interface 32 to communicate with the host monitor system via the communication lines 39; and
- an internal bus 31 connects components of the target system 30 to the target processor 34.

In particular, the system hardware 36 can include a real-time clock that performs operations such as those mentioned above.

The user command interface 18 accepts user input. The user first specifies a particular type of the target processor 34 to be used in development and testing. Application specific attribute values and settings can be issued to enable initialisation of the target processor 34. The user interface 18 also displays the responses returned by the target processor 34, informing the user of the results due to the input.

The user interface 18 communicates with the monitor module 15 via user commands 28 and monitor responses 29 when sending or receiving information from the target system 30. The monitor module 16 reads in the set of target specifications 19 and, together with the input from the user interface 18, constructs communication commands 22 to be sent to the communication module.

A collection of standardised, processor independent primitives reside in the communication primitives module 14. The set of primitives provide access to data in the RAM and registers within the target processor 34. A corresponding set of primitives is also needed to carry out the commands. In order to achieve a higher level of encapsulation so that the differences in hardware architecture of different processors can be hidden, the primitives that carry out the commands are implemented differently across various types of processors that can be used as the target processor 34. The primitives are to be named in a similar manner, so that those with the same name are always functionally similar, regardless of the processor platform. Using this approach,

the differences that exist in the architectures of different target processors are hidden from the host system 10.

The target specifications depository 19 contains processor specific information as well as application specific information of the target system 30. Processor specific information includes the register configurations and the RAM and ROM maps of the target processor 34. Application specific information includes a description of RAM allocation and usage. Use of program variable names for similar functions can be standardised across different processor platforms. However, the address locations of similarly named variables can differ.

The auto test module 16 derives its input from a pre-programmed sequence of test operations. The test operations are sent to the monitor module 15 as a test sequence 24. Together with the processor specifications retrieved from the specifications depository 19, the monitor module 15 sends the appropriate commands 22 to the communication module.

The monitor module 15 receives responses 23 from the target system 30 via the communication module and redirects the responses back to the auto test module 16 as a series of test results 25. The auto test module 16 sends the results for storage 27 in the database engine 17. In addition, the auto test module 16 is able to interpret the results that are received from the monitor module 15 and act accordingly. For example, it can choose to terminate the tests if it has detected that the execution of the target processor 34 has entered into an erroneous state and is unable to proceed without human intervention.

The database module 17 is used to tabulate the responses of the target system 30. The database 17 also stores different test sequences needed to test a target system for a particular application. A database record can contain:

- type of application,
- type of processor,
- type of test,
- documentation for test,

- list of commands used.

The database engine 17 sorts the records by fields, or by keywords in the fields.

A communication primitive is a well defined command which denotes the smallest functionality required for communication. For example, a function to receive one single byte from the communication port `<receive_one_byte()>` can be classified as a primitive, if one byte is the word size for a particular processor. In order to receive ten bytes, the `<receive_one_byte()>` command would need to be invoked ten times.

The collection of communication primitives interprets the information received from the communication lines and carries out the appropriate actions. These actions can include the fetching or writing of data to registers or memory of the target system 30.

The code to implement the set of communication primitives in the target system 30 is kept separate from the application firmware 35. The number of primitives is kept to a minimum so as not to unduly interfere with performance of the processor 34 on the target system 30. The complexity of each of the primitives is also kept to a minimum so that they are easily implemented and will not take too many cycles of the processor 34 to execute.

The host processor 11 issues a `<request for service>` signal to the target processor 34 whenever it requires service. The target processor 34 needs to have a means to detect, interpret and service the request that is being sent from the host processor 11. The means to detect the request signal depends on the architecture of the firmware 35 that is being chosen, which varies with different applications. The scheme selected should not introduce interference into the target system 30 being monitored.

A request line from the host processor 11 can thus be connected to one of the interrupt pins of the target processor 34, thereby generating an interrupt signal whenever there is a request from the host processor 11. The priority level of the interrupt assigned to the incoming request needs to be selected so that the host

processor 11 is able to obtain reasonable service from the target processor 34. The servicing of other time critical tasks in the target processor 34 should not be delayed or missed.

Alternatively, a polling strategy can be used, whereby the target processor 34 actively and periodically polls the status of the pin connected to the request line. The architecture of the firmware determines how polling and servicing of requests are carried out. The firmware can be organised into many sub-modules with a main controlling module that calls each submodule in turn. In such a case, the polling and servicing function can be implemented as one of the submodules that is called by the main module. The firmware 35 on the target processor 34 can be also organised into tasks that can be activated with respect to a timer source, in a sequential manner, or when some other wake-up conditions are satisfied. The polling operation for the request signal can be implemented as a single task that is woken periodically up by the scheduler.

The communication protocol for the arrangement of Fig. 1 defines the standard in which the target system 30 and the host monitoring system 10 must adhere to in the exchange of information.

It is important to note that in a given application, the ports of target processor 34 may be almost fully utilised. As such, there may not be many available ports that can be used for communicating with the host monitoring system 10.

If the target system 30 being monitored is a real-time system with critical timing requirements, the communication with the host monitoring system 10 should not affect the ability of the target processor 34 to meet its own timing requirements.

When the communication ports of the target processor 34 are all being used by its external sub-systems, it becomes necessary to multiplex the use of those communication ports. This is illustrated in Fig. 2 and is described in detail later in this specification. Using this method, additional control line(s) are used to arbitrate the allocation of the communication lines.

With the help of the control line(s), the target system 30 is able to detect instances when the host monitoring system 10 is requesting service. The target processor 34 can choose to postpone the servicing of the monitoring system 10 if there is also a request from its own external sub-system. This ensures that any requests from its external sub-system is serviced with minimum or no delay.

Conversely, the monitoring system 10 can also detect requests for service from the target system 30. In response, it relinquishes the communication lines so that the target processor 34 can be involved in the exchange of information with its external sub-systems.

Data transfer between the target processor 34 and the monitoring system 10 can either be in serial or parallel form. For serial transfer, a synchronous transmission scheme is preferred, since such a scheme can be applied to a wider selection of processors when compared to an asynchronous scheme. In the synchronous scheme, a bit clock is sent together with a bitstream, so that the receiver can latch the data as it comes in. For parallel transfer of data, a word clock is required to allow the receiver to latch in a word of data.

A number of possible configurations exists to implement the communication protocol. The choice of configuration depends on the number ports available, the level of error immunity and recovery needed and also the presence of any external sub-systems that is attached to the target processor system 30.

In addition, the monitoring system 10 can be used in conjunction with a device (not illustrated) that analyses the input and output signals of the target processor 34. The signal analysing device captures the logic levels at the pins of the processor 34 and sends a representation of the levels to the host monitoring system 10. The analyser device is also capable of driving appropriate signals onto the input pins of the target processor 34. In this way, the response of the target processor 34 can be monitored from its output pins. The use of such a device in conjunction with the monitoring system 10 allows the user to access both information internal to the embedded processor and also externally at its pins. This integrated test and diagnostics sub-

system has the capability of accessing data from both the pins, as well as the RAM and registers of the target processor 34.

Turning now to Fig. 3, an overview of the universal emulator apparatus 80 can be provided. The system 80 comprises of the following modules:

- a standardised user interface module 90;
- a processor resource data depository module 89;
- a test database engine module 88;
- an auto tester module 87;
- a generic emulator software engine module 86;
- a generic emulator hardware module 83;
- a generic remote processor pod module 85;
- a host processor 81; and
- a host bus 82 interconnecting the modules with the host processor 81.

The standardised user interface module 90 allows the user to communicate with the emulator and the target processor system 30. Unlike conventional emulators, the user is able to view all the memory and register contents of the target processor 34 in real-time, while the processor 34 is executing. In addition, various debugging tools such as the setting of breakpoints in the code and trace functions are available. The outlook and usage of the interface remains the same regardless of the processor 34 chosen. There is no lead time needed for the user to orientate himself to a new environment in the case of a new emulator. The user interface 90 issues and receives commands 98 and data 99 via the generic emulator software engine module 86.

The type of target processor 34 is selected from the processor data depository module 89 to initialise and configure the emulator apparatus 80. It contains the processor category, the input and output pins assignments, the packaging type, the register and memory maps, unique resources that exist within the processor and other definitions to fully define the processor. This module also has the facility to add, edit, search and delete information pertaining to the processors.

The test database engine module 88 tabulates the responses of the target system 30 to the sequences of commands issued by the auto tester module 87. The database 88 also stores the different test sequences needed to verify the functionality of a particular application. A database record can contain:

- type of application;
- type of processor;
- type of test;
- documentation for test; and
- list of commands used.

The test database engine 88 sorts the records by fields, or by keywords in the fields.

The auto tester module 87 obtains its set of command sequences from the test database engine 88. These commands are in turn issued as a test sequence 93 to the generic emulator software engine module 86. The target system's responses to the stream of commands are channelled back to the auto tester module 87 as a series of test results 94.

The ability to handle a variety of processors is made possible by having a generic emulator software engine 86. This is the heart of the whole emulator apparatus 80, and derives its settings and manner of operation from the information extracted from the processor data depository 89. Commands 98 from the user interface module 90 are channelled to the engine 86 for processing. In conjunction with the processor type selected previously, the commands 98 received are translated into a predefined format before they are issued to the generic emulator hardware module 83 as an output bitstream 91. Conversely, input bitstream 92 data acquired from the hardware module 83 are decoded according to the processor type and sent to the user interface module 90 via the data line 99.

The generic emulator hardware module 83 interfaces between the software engine 86 and the remote processor pod 85. A communication strategy is employed

that enables very high speed data transfer along communication lines 84 to allow the viewing of register and memory contents in real-time.

Processors from different families and manufacturers utilise different packaging and pinouts. The general remote processor pod 85 has adaptors that accommodate the distinct packaging and pin configuration of each processor.

Returning to Fig. 1, specific embodiments of the arrangements shown can now be described, wherein the target system 30 is formed by a consumer product such as a CD radio cassette player. The host monitor system 10 generally resides on a personal computer, and the host processor 11 represents the controlling processor of the personal computer which performs the coordination and control of all the various modules that constitute the host monitoring system 10.

The database engine 17 performs all the necessary functions required to document program flaws detected during the process of firmware development and testing of the CD radio cassette player. The remedies and the test sequences to verify the absence of the respective flaws are also stored in the database. A search engine is incorporated to allow the retrieval of information regarding specific program flaws through the use of keywords.

The auto tester module 16 issues sequences of commands to the CD radio cassette player without the need for any human intervention, and derives its input from script files. Each script file implements a particular test to certify a certain function of the CD radio cassette player. This module is heavily used in firmware verification tests involving events which are dependent upon the real-time clock of the player, and also channels all test results to the database module.

The monitor module 15 receives test sequences 24 from the auto tester module 16 or user commands 28 from the user interface 18 and issues application specific instructions 22 to the communication primitives module 14, based on the CD radio cassette player specifications that have been previously chosen. Conversely, responses 23 from the player which are obtained via the communication primitives module 14 are

channelled to the auto tester module 16 as test results 25 and the user interface 18 as monitor responses 29.

Detailed information regarding the different microcontrollers and their memory maps that can be used with the CD radio cassette player are stored in the target system specifications depository 19. A particular microcontroller model is selected prior to the tests. The data concerning that particular microcontroller and the use of its memory is directed to both the monitor 15 and auto tester 16 modules. This method allows the same user commands 20 and test sequences 24 to be used across different microcontroller platforms on the CD radio cassette player.

The user interface 18 is preferably implemented by a graphical user interface (GUI) module 18 employed in generating multiple data views to display the memory and register contents of the player. Modification of the contents is made by simply changing the data displayed in the various views. A script file viewer and editor is also present, to form an integrated monitoring interface. Additional views showing the time and data status of the player are also present. The database module 17 is preferably contained within the user interface. All commands entered by the user are directed to the monitor module 15.

The communication primitives module 14 obtains instructions from the monitor module 15 and translates them into well defined communication primitives. The string of primitives is transferred to the hardware interface in the form of an output bitstream 20. It also receives input bitstreams 21 from the CD radio cassette player via the hardware interface 13, and decodes and interprets the information received and directs them to the monitor module.

The hardware interface 13 is implemented using a printed circuit card which is inserted into an expansion slot of the personal computer. Communication between the card and the communication module is effected by the use of hardware interrupts. The card contains circuits to generate and receive signals from the hardware interface 32 of the CD radio cassette player. Input and output registers are used to buffer the transfer of data between the personal computer and the player.

The communication lines 39 between the host computer system 10 and the CD radio cassette player are shown in detail with reference to Fig. 2.

The coordination and control of all the processes in the player are handled by the microcontroller of the CD radio cassette player, which is represented by the target processor module 34. The design of the player allows for different models of microcontrollers to be used. A change of microcontrollers would require new information to be selected for use in the target specifications module 19.

The various devices that reside on the CD radio cassette player, viz the CD player, tuner, cassette player, display and the real-time clock are depicted by reference numeral 36. These devices are controlled by the microcontroller.

The processor firmware 35 provides operational instructions to the microcontroller 34.

The communication primitives module 33 obtains commands from the host 10 in the form of bitstreams 38 via the hardware interface 32 of the player. It interprets the information received and directs them to the microcontroller 34 via the bus 31 in the form of commands. Data and responses from the microcontroller 34 that are intended for the host monitoring system 10 are channelled to the module 33 for translation into communication primitives and subsequently to form the output bitstream 37.

The hardware interface 32 of the CD radio cassette player functions as a junction and buffer between the communication link 39 and the microcontroller 34.

Fig. 2 shows how the communication, arbitration and interrupt lines can be connected between the host monitoring system 10, the microcontroller 50 (34) of the CD radio cassette player and a CD LSI module 51 of the player.

The CD LSI module 51, occasionally sends CD subcode data 58 to the microcontroller 50, and uses serial in 54 and serial clock lines 54 that constitute the communication link 39.

An arbitration and request line 52 is used to determine the direction of data flow between the host monitoring system 10 and the player. In addition, the line 52

also serves as a means for the host processor 10 to indicate a service request to the target microcontroller 50. By asserting a certain logic level on the arbitration line 52, one party informs the other that it has data to send. Whenever the microcontroller 50 intends to send data to the host system 10, it conducts a check to ascertain whether the arbitration line 52 is asserted. If the line 52 has not been previously asserted by the host system 10, it asserts the line 52 and sends out clock 55 and serial data 53 to the host system 10. The signal level on the arbitration line 52 is also used to control a switch 56 that enables or disables communication between the CD LSI module 51 and the microcontroller 50.

The serial data line 53 transports data bitstreams from the microcontroller 50 of the player to the host monitoring system 10.

A serial data line 54 is provided that transports command and data bitstreams from the host monitoring system 10 to the microcontroller 50 of the player and also CD subcode data 58 from the CD LSI module 51.

A clock signal on the serial clock line 55 is used for synchronous communication, and is always generated by the microcontroller 50 of the player. The use of the line 55 is shared between the host system 10, the microcontroller 50 and the CD LSI module 51. When the microcontroller 50 is notified that the host system 10 is about to send data, it generates clock signals on the serial clock line 55 to retrieve information from the host 10. During this time, the connection between CD LSI module 51 and the serial input port of the microcontroller 50 has to be disabled.

An interrupt line 59 is provided as shown and whose signal originates from the CD LSI module 51. When the CD LSI module 51 requires service from the microcontroller 50, an interrupt request signal is generated. The particular interrupt request signal line 59 is also connected to the host monitoring system 10, thereby allowing it to know that the CD LSI module 51 is about to seize the use of the serial data lines 54,55. If the host monitoring system 10 is in the middle of a transmit or receive operation, it first finishes the transmission or reception of the current word of information, before releasing the use of the line 54 to allow serial communication to

take place between the microcontroller 50 and the CD LSI module 51. Upon receiving the interrupt from the CD LSI module 51, the microcontroller 50 first completes the execution of the current instruction before generating clock signals to retrieve data from the CD LSI module 51. After executing the interrupt service routine, the microcontroller 50 writes a signal onto the arbitration line 52 to inform the host monitoring system 10 that it can resume its previous operation prior to the interrupt.

The switch 56 is voltage controlled and employed to enable and disable the lines connecting the CD LSI module 51 to the communication link 59. When the switch 56 is open, the serial clock 57 and subcode data out 58 lines of the CD LSI module 51 are disconnected, and communication is only possible between the host system 10 and the microcontroller 50. When the switch 56 is closed, the host system 10 disables its communication module and the exchange of data between the CD LSI module 51 and the microcontroller 50 is then possible.

Applying the above exemplary embodiment to Fig. 3, the host processor 81 represents that within a personal computer where the universal emulator apparatus resides on. It performs the coordination and control of all the modules that constitute the emulator.

The test database engine 88 documents program flaws detected during the course of firmware development and testing. The remedies and the test sequences necessary to verify the absence of the respective flaws are also stored in the database. A search engine is incorporated to permit the retrieval of information regarding specific program flaws through the use of keywords.

Automated testing of the firmware is achieved by using the auto tester module 87. The module 87 issues sequences of commands 93 to the target processor 50 without the need for any human intervention. It derives its input from script files. Each script file implements a particular test to verify a certain function of the firmware. This module is heavily used in firmware tests involving events which are dependent upon real-time clocks. All test results are channelled to the test database engine 88.

The generic emulator software engine module 86 forms the core of the universal emulator apparatus 80 and derives its input from both the user interface module 90 in the form of user commands 98, and the processor resource data depository 89 in the form of processor definitions 97. The module 86 processes these two sets of data by mapping the user commands with respect to the resources within the selected processor 50. These processed commands are then sent to the hardware interface 83. Likewise, data received from the hardware interface 83 are decoded and transferred to the user interface module 90. This method allows the emulator to exercise the flexibility that permits it to target a wide variety of processors. Sequences of commands 93 are issued to the software engine 86 from the auto tester module 87. The commands 93 are treated in the same manner as the individual commands received from the user interface module 90.

Information which fully defines the internal resources and characteristics of the variety of target processors 30 available for use, are stored in the processor resource data depository module 89, which has facilities to add, edit, search and retrieve data. The user initialises the hardware and software of the emulator 80 by selecting the appropriate type of processor from the store. The information retrieved is delivered to the generic emulator software engine 86 so as to allow it to translate the user commands and memory maps according to the processor chosen. In addition, the data is used to configure the operation of the generic remote processor pod module 85.

The user interface module 90 is realised as a graphical user interface (GUI). Multiple windows and data views are used to display the contents of the various memory segments and registers in real-time. The data for the different views are obtained from the generic emulator software engine module 86. The contents of the memory can be modified by directly changing the data displayed in the views. These changes are sent as user commands 98 to the generic emulator software engine module 86. The user interface module 90 also implements the code tracing, stepping, and breakpoint features. Selection of the processor type can be made from within the user interface module.

The hardware interface 83 is preferably implemented as a printed circuit card that is inserted into the expansion slot of the personal computer, and contains circuits that enable high speed data exchange with the generic remote processor pod module 85. The communication protocol used features data error detection and correction, for reliable operation. Communication with the generic emulator software engine 86 is effected by using hardware interrupts. Registers are used to buffer incoming and outgoing data.

The communication link 84 between the hardware interface 83 and the generic remote processor pod 85 is designed to have the capacity to handle the bandwidth of the most demanding processor available.

The generic remote processor pod module 85 is generally inserted into the processor socket of the target system 30. Its ability to accommodate a wide variety of processors is made possible by the use of a selection of hardware adaptors that conform to the different packaging and pin requirements of the various processors.

Fig. 4 shows the flow chart of the real-time operating system which is formed about a real-time operating system scheduler 120. The various firmware code segments are grouped according to the specific application function to which they belong. The firmware thus comprises of groups of codes (122,124,126,128,130). The processor 34 attends to the many application functions by employing a time sharing strategy to cycle through the different groups. When the processor 34 approaches a group and finds that no service is required, it then proceeds to attend to the next group in line. Thus, the frequency whereby a group of codes is serviced depends on the total number of groups that the processor 34 has to cycle through. A software solution with too many groups will cause the application to be unresponsive, as the average waiting time for service is increased. A compromise has to be struck that allows an optimal number of program groups with the given timing constraints. This approach to firmware design permits real-time monitoring to take place, with minimum intrusion to the execution of existing application programs.

Communication between the host 11 and the target processor 34 is effected by the communication software group 122 which contains routines to detect and interpret the *<request for service>* signal from the host processor 11. Other routines to read and modify the memory and register areas of the processor are also included within this group.

A CD player software group 124 contains the routines to implement all the necessary CD player functions.

Those software routines required to realise the tuner functions are contained within the tuner software group 126.

The operation of the cassette player is managed by the programs within the cassette player software group 128.

The programs to handle the display of the CD radio cassette player are represented by the display software group 130.

Previous method of validating the firmware are both tedious and time consuming, especially in tests for events related to the real-time clock. With the capability of being able to remotely read and write the target processor's registers and memory manually or automatically through the use of commands in script files, the presently disclosed systems significantly reduce the time needed to test the firmware, or to verify a particular firmware modification. With the use of standardised communication primitives, the same monitoring system can be applied to a wide variety of processors, as long as the primitives are implemented in the processor's firmware. This results in a faster turnaround time for firmware development.

The use of the universal emulator apparatus of the preferred embodiment results in improved cost efficiency by eliminating the need to acquire a collection of conventional processor specific emulators. Productivity is also achieved by using a standardised user interface, rather than having to adapt to the different interfaces employed by different emulators.

The standardisation of variable assignments and label names in the firmware across processors for similar classes of applications reduces the tedium of

comprehending a new set of codes for the same application. The lead time necessary to switch from the use of one processor to another in a particular implementation is thus reduced with this strategy.

The foregoing describes only a number of embodiments of the present invention and modification, obvious to those skilled in the art, can be made thereto without departing from the spirit and scope of the present invention.



CLAIMS

1. A real-time monitoring apparatus for testing and debugging of firmware in an embedded processor based system, said apparatus comprising:

- a host computer sub-system;
- a target sub-system representing the embedded processor system;
- a communication link sub-system interconnecting the host computer sub-system and the target sub-system; and;
- a monitoring sub-system residing on the host computer sub-system for monitoring operation of said target sub-system in real-time and retaining monitored results thereof.

2. Apparatus according to claim 1, wherein said host computer sub-system comprises a host processor interconnected with a user command console via which user commands are input to said host sub-system.

3. Apparatus according to claim 1, wherein said target sub-system comprises an embedded target processor and peripheral apparatus external to said target processor and which are controlled by said target processor.

4. Apparatus according to claim 1, where said communication link sub-system comprises:

- data, clock and control lines interconnecting said host and target sub-systems;
- a communication protocol for establishing data and control transfer between said host and target sub-systems; and
- a set of communication primitives in the target sub-system.

5. Apparatus according to claim 4, wherein said communication lines serve as a minimum and sufficient number of links through which commands, data and control signals are exchanged between said host and target sub-systems.

6. Apparatus according to claim 4, wherein said communication protocol is a predefined or standardised format of commands, data, arbitration and control signals to which said host and target sub-systems adhere to in the exchange of information therebetween.



7. Apparatus according to claim 6, wherein said set of communication primitives residing in the host sub-system interpret signals on the communication lines and handle the receipt and transmission of commands and data according to said protocol.

8. Apparatus according to claim 6, wherein said set of communication primitives in the target sub-system interpret signals on the communication lines and handle the receipt and transmission of commands and data according to said protocol.

9. Apparatus according to claim 1, wherein said monitoring sub-system includes at least:

- a user interface sub-system;
- a target memory monitoring sub-system;
- an auto tester sub-system; and
- a database sub-system.

10. Apparatus according to claim 9, wherein said user interface sub-system accepts input from the user, and displays current operation status and data from the target sub-system.

11. Apparatus according to claim 9, wherein said target memory monitoring sub-system allows the user to query and modify the memory areas of the target sub-system.

12. Apparatus according to claim 9, wherein said auto tester sub-system allows the user to pre-program test sequences or test cases for automatic testing of the target sub-system without intervention by the user, and subsequent reporting of results of the tests and any errors or exceptions identified therein.

13. Apparatus according to claim 9, wherein the said database sub-system documents anomalies together with their possible causes and remedies.

14. Apparatus according to claim 1, further comprising a device that monitors external input and output signals of said target processor, so as to achieve analysis of the operation of the target sub-system.



15. A method for the standardised development of communication primitives to allow one of a plurality of target sub-systems, each having a different controlling processor, to be tested using a host monitoring system, said method comprising the steps of:

(a) standardising variable names within each field of application of said target sub-systems and retaining same within said host monitoring system; and

(b) standardising communication primitives for and amongst each of said different controlling processors and a controlling host processor of said host monitoring system and retaining same within each one of said target sub-systems.

16. A method according to claim 15, wherein the standardising of said variable names imposes similar names for items of similar type, use or function so that a given variable name will always be synonymous across said different processors, though the internal implementations or address locations might differ.

17. A method according to claim 15, wherein the standardising of said communication primitives uses similar names for primitives of similar functions, even though the primitives can be implemented differently across said different processors to achieve the same functionality.

18. A universal in-circuit-emulator apparatus for development and testing of one of a plurality of target systems, each of said target systems having a controlling target processor, said apparatus comprising:

- a user interface sub-system;
- a processor resource data depository sub-system;
- a test database engine sub-system;
- an auto tester sub-system;
- a generic emulator software engine sub-system;
- a generic emulator hardware sub-system; and
- a generic remote processor pod sub-system.

19. Apparatus according to claim 18, wherein said user interface sub-system displays the register and memory resource contents of the target processor in



real-time while the target processor is executing and allows the issuing of commands and modification of memory data of the target system.

20. Apparatus according to claim 18, wherein said standardised user interface sub-system displays register and memory resource contents of the target processor in real-time whilst said target processor is executing, allows the issuing of commands and modification of memory data within said target system, and whose outlook and usage remain the same regardless of the target processor used.

21. Apparatus according to claim 18, wherein said processor resource data depository sub-system contains all that information necessary to fully define each particular one of the target processors, with facilities to add, edit, search, and delete any data pertaining to the target processors.

22. Apparatus according to claim 18, wherein the said database sub-system documents anomalies relating to the target system together with their possible causes and remedies.

23. Apparatus according to claim 18, wherein said auto tester sub-system allows the user to pre-program test sequences or test cases which can be used for automatic testing of the target sub-system without user intervention, followed by the reporting of results of the tests and any error or exceptions.

24. Apparatus according to claim 18, wherein said generic emulator software engine sub-system provides all the necessary functions and logic to implement said apparatus, including the flexibility of being able to configure itself to handle a wide variety of the target processors.

25. Apparatus according to claim 21, wherein said generic emulator hardware sub-system interfaces said emulator software engine with said generic remote processor pod to enable high speed data transfers and error correction necessary for viewing the register and memory contents in real-time.

26. Apparatus according to claim 18, wherein said generic remote processor pod has the ability to be used for a wide variety of the target processors by



having adaptors that accommodate the different packaging and pin configurations of the target processors.