SOFTWARE SIMULATION OF AN IMPLANTABLE PACEMAKER

By R. E. Riley and M. A. Rossing

MEDTRONIC, INC.
P.O. Box 1453
Minneapolis, MN 55440

Abstract

Software simulation of a hardware system has been used as an effective tool in evaluating and testing software for microprocessor-based control systems. This paper reports on a system simulator used in the development and testing of software for an implantable pacemaker. The paper describes the functional and structural characteristics of the simulator, the user interface, run-time information that is provided from a simulation, and experiences with the system simulator.

Introduction

A typical approach to verification and testing of a real-time software application has been to use an emulator board with a host system interface to perform module test execution of code on the target processor, usually at the machine/assembly language level or lower. Instruction level simulators on a host system that model the functions of the target processor have also been used. In both of these approaches, it is usual to test only the compatibility of the software with the target processor. External interrupts and I/O are handled with stubs or drivers. These approaches are limited in their ability to analyze system interaction at an early enough time in the development process to respond to design and logic errors in either system or software design. Integration/system test of the software on the target processor with other components in the system is usually not performed until breadboards or actual hardware builds are available.

A software environment simulator provides additional advantages to assist the software tester in verification and test of real-time software. An environment simulator operating on a host system may include the instruction level simulation of the target processor with a symbol table reference feature and a pin-level (functional) simulation of all other electrical and mechanical components of the system. This would enable software or system testers to perform module and system tests in simulated real time when breadboard systems are unavailable. The environment simulator used on the host system provides consistent duplication of simulated system conditions, flexibility to respond to software and hardware modifications, and the availability of other software support tools on the host (e.g., assemblers, compilers, and test generators).

This paper describes a software environment simulator developed to model an implantable pacemaker which includes a microprocessor component. The Implantable Pacemaker System Simulator (IPSS) is used for module test and system test of software to be embedded in the pacemaker. The paper summarizes the structural and functional characteristics of the model, provides some implementation details, and reports on results and possible future uses of the simulator.

Summary of Requirements

Requirements for the design and implementation of the IPSS include a model of the implantable pacemaker system and an external interface to the simulator. The model must represent the internal structure and edge-level timing of all electrical components of the system and the electrical characteristics of the heart. The external interface must allow a user to specify pacemaker and heart conditions, to interrogate registers and level settings, and to modify parameters as required for testing software operation in the system. IPSS is not intended as a hardware debugging tool, although a sufficient implementation of the model of the hardware will allow some amount of system debugging to occur. Summarized below are some selected requirements for the IPSS for the system model and the user interface.

System Model

The implantable pacemaker system is composed of electrical components, lead wires and electrodes, and the right atrial and ventricular chambers of the heart. There are five electrical components including: 1) a low-power microprocessor with on-chip ROM and RAM with features similar to an Intel 8080; 2) a digital controller with a serial I/O interface, a series of 8-bit timers with selectable clock speeds, a selectable speed clock, and an interrupt controller; 3) a linear/digital interface circuit with a 40 kHz clock, telemetry receive/transmit block, output drivers, and heart activity sensing translation; and 4) a lithium-iodine battery. Lead wires and electrodes for both the atrium and ventricle (dual-chamber system) serve to stimulate the heart and to sense electrical activity in the heart. These components connect the linear/digital circuit to the heart. The stochastic model of cardiac activity for the simulator requires only those human heart functions which may be sensed or stimulated by this dual-chamber system. Electrical specification for all components of the system is required for clock speeds, power consumptions, and timing dependencies. IPSS models the following:

- all external pinouts of the electrical components,
- all internal functional characteristics of the electrical components.
electrical components and the dual-chamber heart with appropriate time delay and signal propagation,
- the delivery of pulsed stimulation to the heart chambers,
- the detection of electrical activity of the heart, and
- the dual-chamber electrical system of the heart including a variety of pathologies such as atrioventricular block and arrhythmias.

User Interface

The functional requirements for the user interface to the IPSS include a method for specifying system conditions (input) and a method for providing information on system operation (output). Possible system conditions input to the simulator include the following:

- loading of the system ROM area
- loading of the processor op code table
- loading of a symbol table for ROM and RAM address mapping
- specifying interconnect of pinouts of the electrical components in the pacemaker to determine/change signal propagation in the system
- specifying energy consumption attributes for static and dynamic current drain in each of the electrical components
- displaying or modifying internal values at any time during the simulation for all digital registers and timers, ROM and RAM memory locations, pinout levels, flip-flops, and heart variables
- processing breakpoint conditions to suspend simulator operation and perform breakpoint actions at a specified time or when a set of condition qualifiers are true during the simulation
- breakpoint actions to perform display and modifications of internal values
- condition qualifiers for Boolean evaluations (equal, not equal, less than, greater than, less than or equal, and greater than or equal) of internal registers, memory, pinouts, or variables, and for edge level changes from high to low or low to high
- termination of the simulation at a specified time
- enable and disable of the breakpoint handler for a specified time frame
- energy consumption statistics for the electrical components and microprocessor instructions executed over a specified time duration
- specifying a string of data to be transmitted to the pacemaker via the telemetry and the serial I/O interface, and
- specifying the following special heart conditions:
  - normal sinus rhythm
  - normal sinus arrhythmia (respiratory modulation)
  - sinus bradycardia
  - sinus tachycardia
  - idioventricular rhythm
  - atrioventricular block (1st block, 2nd Mobitz type I and II block, 3rd block)
  - atrial premature systole (with or without bigeminy)
  - junctional premature systole (with or without retrograde conduction)
  - ventricular premature systole (with or with retrograde conduction)
  - paroxysmal supraventricular tachycardia
  - ventricular tachycardia

The output of the simulator includes the following:

- display of a simulated real-time value (100 nanosecond resolution) for each display request in the simulation
- display of all data requested by the input
- a summary of energy consumption by the system for the simulated time period, and
- contents of the RAM at the termination of the simulation

Implementation

IPSS has been implemented in Pascal on a Prime 750. The simulator requires two user-supplied text files. One file contains both a memory map of the ROM and a symbol table to initialize simulation tables in IPSS. IPSS uses the hex object file created by the microprocessor cross-assembler hosted on the Prime 750 for this input file. The user must have completed the assembly of a program for this file to be available to IPSS. The second input file is a "scenario" file created by the user containing a series of IPSS commands. These commands specify pin interconnections, energy parameters, display requests, modification of values, breakpoint conditions, telemetry transmissions, and heart conditions. IPSS commands are summarized in Table 1, and the general syntax of a command is as follows:

\[ (<\geq> \langle\text{start time}\rangle <\langle\text{duration}\rangle> <\langle\text{command}\rangle <\langle\text{parameters}\rangle\rangle) \]

Where: () indicates optional arguments.

\(<\geq>\) indicates relative time increment from the last command processed in the file.

\(<\text{start time}>\) is a time designation in seconds, milliseconds, or microseconds when the event is to occur.

\(<\text{duration}>\) is a time designation in seconds, milliseconds, or microseconds indicating the length of time the command is in effect (active) in the simulation.

\(<\text{command}>\) is a command name as described in Table 1.

\(<\text{parameters}>\) are arguments specific to the various IPSS commands.

Each command processed by IPSS from the scenario file generates an "event" during the simulation. The simulator is an event-driven model where an event is any change in state of pinouts, flip-flops, registers, timers, clocks, memory contents, or other items for any system component. Each event has an associated start time and finish time. Events are specified by the scenario file or are scheduled during simulated operation as the result of some other event. All events are placed in a dynamic master event queue which is ordered by the \(<\text{start time}>\) attribute of the event. IPSS maintains a
real-time clock (simulated, to 100 nanosecond resolution) and processes one event in sequence as the clock reaches the event's start time. This processing loop continues until a termination (STOP) event halts operation of IPSS.

As each event is processed, required actions are performed and new events may be placed in the queue. For example, if a 1 kHz flip-flop were to be simulated, an event would be generated to set the flip-flop to 1 at some start time. When that event is processed, a new event is placed in the queue to set the flip-flop to 0 after 500 microseconds from the current start time of the event. When this new event is serviced, the flip-flop is set to the value of 1 and another event is placed on the queue to set the flip-flop to 0 after 500 microseconds. This sequence, when repeated throughout the simulation, will model a 1 kHz square wave with a trailing edge every 1,000 microseconds.

An event-driven interconnect will model signal propagation throughout the pacemaker system. Events on edge-driven pinouts cause other events to be queued for all pins connected to that pinout. For example, if the microprocessor RESET pinout was connected to the digital controller RESET pinout, then when a reset was initiated on the controller pin, actions associated with the digital controller reset would be performed and an event for reset of the microprocessor queued.

As the simulation progresses through simulated real time, an output file is written with all DISPLAY requests from the scenario file. This file is then evaluated by the user to determine pacemaker system response to the designated conditions of heart activity, component interconnect, and software executed in the simulation. Virtually all system parameters may be examined by careful selection of BRK (breakpoint) conditions and DISPLAY specifications. With every display from the simulator, a corresponding time value is supplied indicating with 100 nanosecond resolution when the condition associated with the display occurred in the simulation. This resolution allows the user great flexibility in observing conditions from one tick of a system clock to the next, from execution of one microprocessor instruction to the next, or from one beat of the heart to the next.

**Experience With IPSS**

IPSS has been used by software engineers for module test and functional test of software to be embedded in the pacemaker system. Tests using IPSS to verify hardware design of electrical components were only performed to the extent necessary to determine that functional blocks of the components accurately reflected timing and control intentions of the hardware design. No formal system test was performed using IPSS to verify hardware design.

Software module tests were constructed by selecting the module, unit, or frame of code to be tested, assigning appropriate values to all registers and memory locations, and setting the microprocessor instruction pointer to the ROM location to begin the simulation. The scenario file input to IPSS provided a good medium for specifying inputs, defining expected results, and displaying actual simulated execution data. Although the entire pacemaker system was available for module test simulations, only the microprocessor and the digital controller were extensively exercised. This is consistent with other methods of module test since it is primarily intended to demonstrate that small portions of code execute on the microprocessor as detailed designs specify. IPSS allowed for complete code execution traces to observe changes to registers, timers, and memory contents. The symbol table reference capability of IPSS was especially convenient during this phase of test to display or modify memory locations of the ROM or RAM areas.

Software functional tests were constructed by selecting heart conditions, electrical component states, and memory parameter contents prior to running the simulation. These functional tests simulated normal system response to electrical activity of the heart model while verifying functional requirements of the pacemaker system. All pacemaker system components were used extensively by these functional tests. This particular pacemaker implementation was designed to be completely software controlled so functional tests of software essentially modeled complete system interaction. IPSS allowed for a wide range of useful system information to be gathered. For example, an IPSS simulation could determine the time (in microseconds) it took to reach a software module in response to a heart event given all the simulated component delays and interrupt control constraints. IPSS also provided system information on interrupt contention and executive control of software flow.

IPSS was valuable in identifying several design incompatibilities for internal system timing response and resolution. These would have been difficult to identify, demonstrate, and correct using conventional breadboard debugging techniques. Since IPSS was available on the same host system that other software development tools were available, modifications to embedded software could be quickly completed, assembled, and retested. Modifications to hardware designs could also be rapidly reflected in the source code for IPSS and IPSS recompiled. Tests with the modified pacemaker model then could be repeated and verified.

**Discussion**

IPSS was developed and used in the absence of a breadboard system for the pacemaker. It was found to be a valuable and reliable tool for verifying and testing software for the pacemaker at both the software module and software system test levels. Simulating all the pacemaker components provided software engineers with a model of the hardware and a means of evaluating software from both a software design and a system design perspective. Software engineers testing software functionality performed many system test engineering tasks while evaluating simulations. The environment simulator does not alleviate the need to test software or perform system verification tests on a breadboard or prototype system when these tools are available. The environment simulator has its greatest advantage in providing an additional level of verification of system execution. Other advantages include providing an early definition of functional system tests (which could also be used in defining breadboard functional tests), and executing in a flexible environment (the host system) where modifications to software and simulated hardware can be quickly performed.

Future applications for environment simulators of implantable pacemakers are currently in the planning stage.
stages. IPSS was designed to easily accommodate additions and deletions of electrical components of the model with minimum impact on the event-driven design. Each electrical component was implemented as a separate program module in the IPSS design so a program library of modules representing an electrical component "chip-set" could be established. By combining (compiling) a core event-driven module with selected electrical component modules, a different environment simulator, modeling a different pacemaker, could then be constructed. The CONNECT command of the simulator could then be used in a scenario file to define the interconnect of the components. This type of development would allow access to testing pacemaker functionality prior to availability of breadboard models.

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT</td>
<td>Specifies names of pins to be connected between electrical components of the system.</td>
</tr>
<tr>
<td>SET CURRENT</td>
<td>Specifies values to be used in energy consumption computations.</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Specifies alternate text files on the Prime to be read as if they were part of the scenario file.</td>
</tr>
<tr>
<td>DUMP</td>
<td>Specifies a text file name to which simulation output is written.</td>
</tr>
<tr>
<td>BRK</td>
<td>Specifies a breakpoint block for the simulation. Boolean conditions are supplied as parameters to indicate whether/when the breakpoint should occur.</td>
</tr>
<tr>
<td>END</td>
<td>Specifies the end of a breakpoint block.</td>
</tr>
<tr>
<td>DISPLAY</td>
<td>Specifies a string to be written to the output file. The string may consist of text enclosed in quotes, values for pins, registers, timers, or memory, or any combination of the two.</td>
</tr>
<tr>
<td>SET</td>
<td>Specifies a value to be assigned to a pin, register, timer, memory location, or other variable in the model.</td>
</tr>
<tr>
<td>STATS</td>
<td>Specifies that energy consumption statistics should be gathered and displayed in the text output file throughout the entire simulation.</td>
</tr>
<tr>
<td>STOP</td>
<td>Specifies the termination of the simulation.</td>
</tr>
<tr>
<td>DBRK, EBRK</td>
<td>Specifies the disable or enable of processing BRK conditions, respectively.</td>
</tr>
<tr>
<td>IPGRCV</td>
<td>Specifies a data string in decimal, hex, octal, or binary to be &quot;transmitted&quot; to the telemetry function of the pacemaker.</td>
</tr>
<tr>
<td>SINUS</td>
<td>Specifies the nominal rate of the heart.</td>
</tr>
<tr>
<td>PR</td>
<td>Specifies the interval between the atrial contraction and the ventricular contraction (PR interval).</td>
</tr>
<tr>
<td>IDIO</td>
<td>Specifies the spontaneous ventricular rate.</td>
</tr>
<tr>
<td>DEG1</td>
<td>Specifies a PR interval to emulate first degree block behavior of the heart.</td>
</tr>
<tr>
<td>MOB1</td>
<td>Specifies that the heart model should exhibit the Wenckebach phenomenon.</td>
</tr>
<tr>
<td>MOB2</td>
<td>Specifies that the heart model should exhibit Mobitz type II block.</td>
</tr>
<tr>
<td>DEG3</td>
<td>Specifies that the heart model should exhibit third degree block.</td>
</tr>
<tr>
<td>APS</td>
<td>Specifies that the heart model should produce atrial premature contractions.</td>
</tr>
<tr>
<td>JPS</td>
<td>Specifies that the heart model should produce junctional premature systoles.</td>
</tr>
<tr>
<td>VPS</td>
<td>Specifies that the heart model should produce ventricular premature contractions.</td>
</tr>
<tr>
<td>SVT</td>
<td>Specifies that the heart model should exhibit supraventricular tachycardia.</td>
</tr>
<tr>
<td>FLT</td>
<td>Specifies that the heart model should exhibit atrial flutter.</td>
</tr>
<tr>
<td>VT</td>
<td>Specifies that the heart model should exhibit ventricular tachycardia.</td>
</tr>
<tr>
<td>QT</td>
<td>Specifies that the heart model should calculate the QT interval, based on the SINOUS, IDIO, or VT.</td>
</tr>
<tr>
<td>RWAVE</td>
<td>Specifies that the heart model should generate a ventricular sense event after each ventricular pacing pulse.</td>
</tr>
</tbody>
</table>

References