SSCI 125 EMULATION AT POINT LEPREAU G.S.

Macey MacLean, Jim Hogg, Hubert Newman, Herbert Thompson
New Brunswick Power, Point Lepreau Nuclear Generating Station
Point Lepreau, New Brunswick, Canada, E0G 2H0

INTRODUCTION

A C language program has been developed at the Point Lepreau Nuclear Generating Station which emulates the actions of an SSCI 125 computer's instruction set and also emulates the actions of those input and output devices (including a RAMTEK display system) which belong to the plant control computer of the nuclear power plant's full scope training simulator. This enables the same DCC software to run on both the $20 million dollar training simulator and an equivalent desktop simulator whose hardware cost is now $16,000 or less. It facilitates making additional copies of the $20 million dollar training simulator for less than 0.1% of the cost per copy.

When this emulation program is run on a 266 MHz DEC single CPU Alpha computer, it simulates the running of the training simulator plant control DCC (digital control computer) at approximately the rate of real time: either faster or slightly slower depending on the compiler options employed and depending on the choice of the modeled machine cycle speed of the SSCI. Using the standard machine cycle speed of 660 nanoseconds, using the emulator fully optimized, and using plant process modeling without optimization the speed is 90% as fast as real time - this is the normal use situation for simulator modelers who were the originally intended users of the emulator. If the modeled SSCI machine cycle speed is slowed down to 792 nanoseconds (making use of spare capacity on the real SSCI to reduce the workload of emulation), and all objects are compiled as optimized, then it takes only about 20 minutes of clock time to simulate over 30 minutes of modeling time. This mode of simulation may be useful for somebody who is not doing emulator or modeling code development. This could include preparing lesson plans for the training simulator, developing or evaluating plant operating procedures, or trying out revised DCC software.

The design of the emulator is described highlighting aspects of its design which distinguish it from previous DCC emulators known to the authors (2),(3). There is a separate companion paper (1) describing the Point Lepreau Desktop Simulator in its entirety.

- This emulator knows the elapsed time of each SSCI instruction, so it can be conveniently run either in a real time simulation context or in the context of non-real time variable step length plant process simulations run on a shared access mainframe computer in company with non-simulation users. This is significant since most nuclear power plant simulation studies are not done in real time.

- This emulator employs the virtual memory mapping and 4 state memory protection scheme of the SSCI 125 which is more computationally expensive than previous Varian emulators. Yet good overall running efficiency is achieved for the emulator because of design and coding techniques that are described.

MOTIVATION

In the past it has been difficult to study the power plant dynamics of CANDU nuclear plants using the plant control computer software. Using a real nuclear power plant for experimentation poses obvious risks and expense. Alternatively one can:

1. obtain full scope training simulator access where the training simulator has a duplicate of the plant control computer. This approach is very useful because of the scope of the training simulator modeling and the fidelity obtained by running the real plant control programs. However, this is no solution if one does not have a full scope training simulator or one lacks sufficient access to such a facility. Full scope training simulators have very high cost and very high utilization for training.

2. One can build or obtain duplicate control computer hardware and interfacing which is also expensive - one goes some fraction of the way in the direction of a duplicate training simulator with some fraction of those large costs. Or

3. one can replace the old assembly language software of the plant control computer with new programs which run on cheap modern hardware. These software emulation programs are supposed to behave like the *programs* that run on the plant control computer. Not only is the initial cost of this approach high, but there becomes an ongoing necessity to modify and check out the software emulation programs when the assembly language programs in the plant continue to be modified. And there are limits as to how much of the plant control computer one might pay to translate.

A fourth approach is hardware emulation. Instead of imitating the *programs* that run on the plant control computers, one writes a program which imitates the plant control computer *hardware* itself. The hardware emulation program is written in a language that allows the emulation program to be moved easily from one computer to another, which allows the programs of the plant control computer to run on any computer where the hardware emulation program is moved.

The annunciation messages from the plant control computer during plant transients are utterly vital for many purposes, and yet these messages are produced by many different plant control and annunciation programs. So it is only with hardware emulation that it is practical to achieve the same extent of annunciation information as in the real power plant. The cost of doing so through software emulation is impractical, particularly because annunciation is one of the most rapidly changing and heavily maintained areas of plant computer software.

Hardware emulation is much simpler than software emulation. Point Lepreau has over 12 linear feet of assembly language program listings for the plant control computers, so translating and maintaining even a fraction of these programs into a portable higher level language is a big undertaking. By comparison there are mere hundreds of distinct machine instructions which require to be handled by a hardware emulator. In addition, hardware emulators require much less maintenance than software emulators because the programs of the plant control computer get changed much more frequently than the hardware is changed.

One of the main barriers to hardware emulation has traditionally been the cost of computing power. Now that massive computing power is available relatively inexpensively, the practicality of hardware emulation has increased considerably, at least when there is a way to overcome the second great problem of hardware emulation - adequately knowing the behaviour of the machine being emulated.

The SSCI 125 is a clone of the Varian V70 series of minicomputers which were very successful in the 1970's. The Varian V70's series were chosen for control of the early CANDU nuclear power plants and for that purpose Atomic Energy of Canada developed the early CANDU executive DCC (plant Digital Control Computer) software and various application programs for plant control, annunciation and data logging.

When the Point Lepreau plant was constructed from 1975 to 1982 it was provided with duplicate Varian V73 control computers designated DCCX and DCCY. Similar DCC's were provided to Wolsung 1 in Korea, Gentilly 2 in Quebec, and the Embalse plant in Argentina. At least a dozen of Ontario Hydro's nuclear units have similar DCC's.

The plant control and executive software for all these plants is written in Varian assembler. The software was extraordinarily costly to develop as it required nuclear plant transients to help in debugging it. So this software has been retained over the years despite the ongoing march of software technology that has made assembly language programming an historical curiosity in most contexts. The assembly language programs that have been painstakingly refined over the years at the Point Lepreau plant are beautiful, and elegantly laid out - not second class software by any proper definition.

When full scope training simulators were built for these power plants, a faithful replication of the plant control computer functions usually meant including a duplicate of the DCC in the simulator with the DCC executive software being highly modified to provide the added simulator functions not required or appropriate for the nuclear power plant. The executive software of the simulator runs application programs effectively identical to the application programs running in the power plant. The same application program images that run in the Lepreau plant control computers also run on the Lepreau simulator, although on some simulators the code is more highly modified because the simulator executive interface to the application programs differs from the interface seen by the same application programs running in the plant.

When the Point Lepreau training simulator was built from 1988 to 1991 it was no longer possible to buy Varian hardware, but the SSCI company built and sold "clones" of the V70's, an early model of which was the SSCI 125. An SSCI 125 was purchased for the Point Lepreau training simulator by CAE, the simulator manufacturer. The Lepreau training simulator has only one control computer designated DCCS which is set up to imitate DCCX in the plant.

The V70's only had an address space of 32k of 16 bit words, which was fine for their time. The code which controls the Lepreau plant is written for this address space, making extensive use of an overlay area where code is loaded in from the multi-megabyte bulk memory unit (BMU). The SSCI 125 has a larger address space consisting of 16 memory maps each of which can address 32K 16 bit words. The DCC executive software for the Lepreau simulator was even more highly modified than is usual as compared to the executive software of the corresponding plant DCC's In the case of the Lepreau simulator, the job of writing this executive software was more of a challenge than usual because the simulator and the power plant were based on two different kinds of computers - one with memory mapping and one without, and the memory protection schemes of the two kinds of computer are also different.

More recently built CANDU power plants in Romania and Korea are using the SSCI family of Varian clones to control both their plants and simulators. The training simulator DCC to be built for the new Chinese CANDU plant remains to be seen. The Chinese plant is to be controlled by SSCI's.

A possible use of the SSCI emulator would be to displace DCC hardware in a new CANDU simulator. It would be easy to build simulator specific functions such as the ability to freeze and restore directly into the emulator itself - this is especially obvious when trying to simulate the failure of DCC hardware components. By building these abilities into the emulator, one could then run almost the same DCC executive software on the training simulator as in the power plant. It would largely remove the need to create and maintain a separate DCC executive for the training simulator.

KEEPING TRACK OF EMULATED TIME

The Basic Approach to Time

The Point Lepreau SSCI and its peripheral devices is modeled as a single sequential process. This achieves adequate accuracy for purposes of overall plant simulation. This is possible because the SSCI is a single processor machine and because of the limited intelligence of the peripherals employed with it. If there were substantial distributed "brains" one would need to model events occurring in multiple time streams and suffer the overhead of coordinating these time streams - thankfully that has not been necessary.

The design objective for simple and reliable time simulation in the Point Lepreau emulator is to try to code the emulator using only two types of time information: the time delay before some future hardware event occurs; or the time it takes for an SSCI machine instruction to execute. Each of these two types of time information is kept in a place that is simple to find and administer as follows.

1. All information on the delays before future hardware events is contained in a single chronologically ordered event queue, where an event is a simple data structure. Each hardware event knows how many nanoseconds it comes after the event before it in the queue. When it is time for an event to occur, it is made to happen by calling the function pointed to from that particular event data structure.

2. Each SSCI machine instruction knows how long it takes to execute. All information on the timing behaviour of that instruction is contained within the function that emulates that instruction. For example, an ADD instruction with direct addressing has a duration of 1320 nanoseconds in the emulator. With indirect addressing the same instruction is modeled to take 1320 nanoseconds plus 660 nanoseconds for each level of indirect.

Time is advanced in the model simply by having each SSCI instruction decrement the time to the occurrence of the next (i.e. first) hardware event in the event queue. Thus a direct addressing ADD instruction decrements the time to the next hardware event by 1320 nanoseconds.

Keeping time information in very few places was felt to be good simulation software design, and trying to rely as much as possible on relative time rather than absolute time was felt also to be desirable for simulation software.

This design objective was achieved with only one exception. The exception was necessary because countdown registers[1] keep on counting down after they time out and eventually get read after they have timed out. Thus one has to keep track of this kind of event for indefinitely long periods after it occurs. This was done by creating an absolute timer and using certain fields in the data structure of the countdown registers for necessary bookkeeping. The use of the absolute timer was limited to the countdown registers to respect the administrative objectives of the original design. The timer is maintained by a recurring 1 millisecond event on the hardware event queue: 1 millisecond (i.e. the time for perhaps 1000 machine instructions) is the timing resolution of the countdown registers.

The basis for estimating the execution times of our individual SSCI instructions is to take the corresponding published times for the Varian V77 machine - which is only approximately true of the SSCI. There are no published times for the SSCI itself. Published times are approximate because in some situations the execution speed depends on the operands, sometimes in a manner more complex than any reasonable person would want to model.

---

[1] Countdown registers are external timers to the SSCI which are wired into different lines of the external interrupt system. The CANDU DCC executive software talks to these different timers to manage different classes of interrupts which are required to exhibit timed behaviour.

Reasonable guesses or experimentally determined times were used for instructions unique to the SSCI. Such a rough approach to instruction timing has no adverse implications to fidelity of simulation, for reasons explained below.

Plant Control Insensitive to Computer Clock Speed (within limits!)

In a well designed real time control application such as a CANDU DCC, the execution times of individual control computer instructions does not impact on plant control dynamics.

All plant control actions occur at fixed points in time which depend only upon devices external to the SSCI. If there is a turbine trip in the plant, that has an immediate impact on the timing of control actions because of a turbine trip interrupt, but the impact arises from outside the SSCI. When the heat transport control program runs every 2 seconds on the SSCI, that arises due to an interrupt originating from the slow function countdown register external to the SSCI - the timing of the interrupt is independent of the speed of the execution of SSCI instructions. Because all plant control actions occur only at points in time fixed externally from the SSCI, it does not matter how fast or slow is the execution of the SSCI instructions, so long as the SSCI is fast enough to squeeze in all the work it has to do according to the externally driven schedule[2].

If the plant control computer is not fast enough to achieve the externally driven schedule, the problem is that the machine has run out of "spare time". But in practice, it is important that there be substantial "spare time" on the plant control computers - or else the computer is not assured of being able to handle a plant transient or be as responsive to operator input as was the design intent. During "spare time" the DCC executive software is executing at its lowest priority level, cycling around the "background executive loop" where various deferrable tasks are done.

The practical consequence of all of this is that the instruction execution times of the emulator do not have to be highly accurate for plant simulation purposes, so long as the times are not overestimated badly enough to remove all the spare time that would otherwise be available. If the time is underestimated that it takes to execute individual SSCI instructions, the consequence to modeling fidelity may only be that the emulator spends a higher fraction of its time spinning around the background executive loop, and a lower fraction of its time executing plant control programs such as heat transport control which are only run once every 2 seconds, regardless of how fast the computer is and regardless of how many times it might be possible to run that program in 2 seconds given more computing power.

The consequence of underestimating the time it takes to execute the SSCI instructions is primarily[3] to waste the computing power of the computer on which the emulator is running - the emulator will take more time on the computer hosting the emulator to simulate 1 second of time of the computer being emulated.

---

[2] The times for the program schedules are data used by the DCC executive software - these data affect the times that the executive software communicate to the various external timers. But the actions of the timers depend on the fixed data values, not on the speed of execution of the DCC software itself.

[3] The executive software has some built in timing tests (such as for RAMTEK IOBIC DMA data transfers) based on counting the number of cycles around a timing loop and requiring an interrupt before a maximum acceptable count is reached. Shortening up the time too severely in the emulator for the particular machine instructions in these timing loops can therefore cause the executive to leave insufficient time for certain data transfers, unless the executive is patched (not done!).

The foregoing arguments have been validated with our emulator by running a reactor trip event using different instruction set timings, as indicated by the following test cases:

1. the instruction timings were the standard timings of the emulator,

2. the time of a basic machine cycle was increased by 20% in the emulator,

3. the machine cycle time was lowered to 95% its normal value, and

4. the normal execution times of some arithmetic instructions were increased by a factor of 10.

In all cases, the plant transient simulation results were "the same" because there was always spare time for the executive software to just spin around the background executive loop. The case where the machine cycle time was increased 20% from 660 nanoseconds to 792 nanoseconds was accompanied by a consequent increase in the speed of emulation, as less computing power for emulation gets wasted in the executive background loop.

More details on validation of the emulator are given reference (1).

Comparison to Time Simulation on Earlier Emulators

All earlier DCC emulators, of which the authors are aware, had no internal sense of time within the emulator itself. These programs were mostly written in Intel 8088 assembler for purposes of limited testing of DCC software, given that Intel based PC's were easier to access than Varian V70's. Plant simulation was not the original objective, so there was no need to model the passage of time. Later when these emulators were adapted to work in a simulation context, it was easiest not to modify the emulator logic, but instead to provide the emulator with a sense of time by using computer hardware timers.

Regardless of the details of how hardware timers are used to give a "time unknowing" emulator a sense of time, it tends to be done using a fixed ratio between time as measured by the hardware timer and time as seen in the modeled process. There is a fixed relationship between time as seen on the wall clock and time in the modeled process. Having a fixed ratio relationship like this, whether it is 1 to 1, or 1 to 2 ,or 2 to 1 makes it difficult to run such an emulator in a time shared context. However, time sharing is the norm on large mainframe computers used in many design and safety analysis simulations. Time sharing is a problem because generally no user is guaranteed, or wants to be limited to, a fixed fraction of computer CPU capacity.

Another potential problem with "time unknowing" emulators is that the timing resolution available from many hardware timers can be crude compared to what one would like. In contrast, there is no ultimate limit to the accuracy of timing possible using the approach employed by the NB Power emulator - it is only a matter of how much one cares.

The Point Lepreau emulator has a built in sense of time in the software itself, so there is no dependence on any timing hardware of the machine on which the Lepreau emulator is run. Therefore this emulator can be run as easily in a timesharing context as in the context of a dedicated use computer. In addition, the Lepreau emulator does not have to waste computer capacity in non-real time applications because it does not need the spare time required by a "time unknowing" emulator which must at times wait for its next hardware clock tick before the emulator can advance time. In a real time application the two different types of emulator would be comparable because both have to wait to be synchronized with a hardware timer.

As previously explained, wasted capacity in the Lepreau emulator can arise from the amount of time spent simulating time in the background executive loop (this is a source of wasted computer capacity in any kind of emulator because it imitates waste in the real world). The waste can be minimized in the Lepreau emulator by experimentally increasing the instruction execution times until just before fidelity of plant dynamics is affected. However, it is better to have enough of a spare time cushion so you can be sure plant dynamics are accurate in your simulation. The same kind of fidelity problems can arise in "time unknowing" emulators that are too aggressive in trying to achieve fast simulation results.

MEMORY MODEL REQUIREMENTS

Description of SSCI Mapping

Throughout this document octal numbers are always preceded by a 0, and decimal numbers are not. The system utilities of Varian V70's and their SSCI clones are octal oriented.

The SSCI 125 has *physical pages* 0 to 0377 where each page is 01000 words (of 16 bits). The memory limit of the Lepreau machine is 512k words, which is a range of 0 to 01777 physical pages limited by the physical memory installed, not the architecture.

The physical pages of the SSCI 125 architecture numbers 0 to 0377. A *physical address* is composed of 3 octal digits to give the offset within the page, and this offset is preceded by the three digit octal page number. Thus physical address 0265125 is offset 0125 words into physical page 0265.

There are 020 maps within the SSCI. Each map consists of 0100 pointers to physical pages of normal RAM memory. The pointers themselves are stored apart from normal memory. Each pointer can have any value from 0 to 0377. Each map therefore defines a complete virtual memory space of 0100 pages of 01000 words: that is virtual addresses 0 to 077777 (the address range of the old Varian V70 machines). With most computers and operating systems, the virtual memory space is larger than the physical memory space, but with the SSCI, the opposite is true!

The SSCI has a program status word (PSW) register which contains important information indicating the current utilization of the different possible maps.

The MAPAC (map active) bit of the PSW is bit 4. If this bit is not set, then no mapping is used and any address referenced in the software has identically the same physical address. In other words one can only address the first 0 to 077777 physical words of memory if MAPAC is 0. Also there are no memory protect violation interrupts unless MAPAC is 1.

The USRMD (user mode) bit of the PSW is bit 5. If this bit is not set, then only map 0 is used and there can be no memory protect violation resulting from the use of privileged instructions (I/O instructions). When USRMD is set, execution of any privileged instruction causes a memory protect violation.

The KEY portion of the PSW is bits 0 to 3. Its value points to which of the 16 maps used when both MAPAC and USRMD bits are set. When USRMD is not set the key bits are not used, except by some SSCI special instructions that never existed on a Varian computer.

As an example of address interpretation consider software address 043561.

If MAPAC is 0, then the physical address is also 043561 - that is offset 0561 words into physical page 043.

If MAPAC is 1, then mapping is as described in the rest of this example.

If USRMD is 0, then map 0 is used . Pointer 043 in map 0 is examined in the hardware and it points to physical page 0311(for example) which means the physical address is 0311561 corresponding to virtual address 043561.

If USRMD is 1, then the map used is determined from KEY. If KEY has a value of 010 (for example). Pointer 043 in map 010 points to physical page 021(for example). So the physical address is 021561 corresponding to virtual address 043561 in this case.

Programming of Mapping and Protection

The problem is to be able to translate virtual addresses in the range 0 to 077777 into a correct physical address in the range 0 to 0377777.

The approach to solving all programming problems in the Lepreau emulator has been to look for speed, and to be prepared to use abundant cheap modern memory to facilitate that objective.

There are 021 cases to deal with: using one of the maps 0 to 017 or using no mapping at all. For each of these cases one constructs an array of 0100000 elements whose cells numbered 0 to 077777. Each cell contains the physical address for the virtual address agreeing with the cell number. Thus cell 2 contains the physical address for a virtual address of 2. One switches between using different sets of 0100000 cells according to the current values of the MAPAC, USRMD and KEY bits of the PSW.

The SSCI 125 software at Point Lepreau only uses 5 maps, and advantage was taken of this in the implementation of the emulator to reduce the appetite for memory for all those extra cells.

The case for no mapping is set up as a pseudo map at compile time: so there are maps 0 to 017 (really only 0 to 4 in our special case) and a pseudo-map. Every virtual address in the pseudo-map has the same physical address. The cells for all the other maps are set up based on the execution of SSCI instructions used in the DCC executive software to set up the mapping on the real hardware. So if a particular map sets a particular virtual page to point to a particular physical page, fully 01000 cells have to be set up as a result in the emulator. So the execution time in the emulator of the instructions to set up the maps is slow. These instructions are only used by the DCC INIT program, so no problem arises from this slowness for the Lepreau simulator application.

These 021 sets of indexing cells are arranged into a single contiguous array ***memmap*** , and a pointer called ***umapbase*** can be computed so that in every case

physical address = ***memmap*** [***umapbase*** + virtual address]

umapbase points to a particular subset of 0100000 cells in the ***memmap*** array which ensures the mapping used is the correct mapping for the current PSW. Every time one of the first 5 bits of the PSW changes, the value for ***umapbase*** is set accordingly. This is a frequent occurrence, but quick and simple.
Memory protection is organized in exact correspondence to mapping: specifying the current map and current virtual page is necessary and sufficient to define the memory protection characteristics of the current virtual address. So one can look up the memory protection characteristics of the current address in a manner exactly analogous to looking up a physical address for a virtual address.

So

memory protect status = ***memprot*** [***umapbase*** + virtual address]

where memprot is an array set up primarily as a result of the SSCI instructions that set the memory protection in each virtual page. A portion of the memprot array is set up a compile time: there is full access throughout the pseudo map where mapping is disabled (i.e. physical address equals virtual address).

Note than when one updates the value of ***umapbase***, it simultaneously serves the needs of memory mapping and protection.

The slowness in initially setting up the maps and the extensive memory used in the emulator for all the pointer cells is more than paid for by the resultant speed in handling both mapping and memory protection in normal running. It becomes fast and simple to look up the physical address or protection of any virtual address.

The most surprising problem in memory addressing encountered in the development of the emulator was a reliance on an undocumented address wrap around characteristic of the original Varian hardware that has been duplicated in the SSCI. When indexing operations are performed on virtual addresses it is possible for the resultant address to exceed the maximum virtual address of 077777. What the hardware does is "and off" bit 15 so that an address like 0102347 becomes 02347. The original feeling was that anyone who used this feature of the hardware deserved to "hit the wall", and should welcome doing so. The emulator was initially written not to "and off" bit 15. However, significant pieces of the DCC software written in the 1970's used this feature of the hardware and so it was necessary to more properly imitate the hardware. Thankfully the adjustment to the emulator was easy.

Memory Protect Violations

There are 2 bits which define the protection status of any virtual page in the SSCI: according to how there bits are set the page is either

- unassigned (usually there is no physical memory for such a page),
- read only,
- read only operand (more restrictive than read only - must not be an executable instruction), or
- full access.

There are three bits in the PSW which define what kind of memory protect violations can occur: the JMPERR bit (bit 11) and the MP bits (bits 9 and 10). The JMPERR bit is set according to whether the violation occurred during execution of a jump type instruction, and the MP bits denote either

- privileged instruction violation,
- instruction fetch error,
- write error, or
- unassigned memory error.

Privileged instruction violations occur as a result of executing "privileged instructions" (a subset of the I/O instructions) when the USRMD bit is set in the PSW.

Instruction fetch errors occur when the SSCI attempts to execute an instruction stored in read only operand memory.

Write errors occur when an attempt is made to modify a word in memory which has either read only or read only operand protection.

Unassigned memory errors occur when an attempt is made to use unassigned memory.

When any of these errors occur, information which is particular to the violation is recorded by the SSCI hardware in addresses 062 and 022. Since the Lepreau SSCI executive uses jump and mark emulation for the memory protect violation handler, the other key piece of information is the return link address recorded by the hardware at the address pointed to by the contents of address 021 in memory.

Although the definitions of the different types of errors seem simple, the memory protect behaviour of the SSCI is complex and quirky. Typically there are multiple words of memory to examine when an instruction is executed, and if there is a violation on one word, there may be a violation on more than one. So there is a question of what happens in the case of multiple violations.

The most common memory protect violation that occurs is intentional in the executive software. The typical call for an executive service from a CANDU DCC application program involves setting up the B register with a specific key value and then creating an intentional memory protect violation by doing a jump and mark instruction into protected memory. On the SSCI the jump and mark is indirect through a memory word that is read only operand (a violation) to a mark word that is also read only operand (a violation) and a jump target that is also read only operand (a violation). The result of all this is not an instruction fetch error for the instruction at the jump target and not a write error for the mark word, but rather an unassigned memory error in the MP bits of the PSW and the JMPERR bit is also set.

The memory protect violation handler of the executive software checks to see if the B register contains the required key, and the call is allowed to proceed. Otherwise the violation causes a program failure or restart of the DCC, depending on the program causing the violation.

As another example of quirkiness, the SSCI can look down a chain of indirect addresses "through" unassigned memory that physically exists to return information about what is at the end of the chain of indirects.

The Point Lepreau emulator does not try to cope with the full complexity of the memory protect behaviour.

There was no attempt to incorporate the information recorded in memory address 022 upon the occurrence of a memory protect violation: the use of this information is only for debugging, and the emulator being a software device has inherently greater debugging flexibility than the hardware ever could.

It was felt unnecessary to deal in the emulator with the ability of the SSCI to look through unassigned memory that physically exists. So as a matter of policy the Lepreau executive SSCI software ensures that the only unassigned memory is memory which does not physically exist. This is not seen as a practical constraint.

In general, there was no attempt made to deal with correct emulation of the simultaneous presence of multiple memory protect violations involving the same SSCI instruction, except for those specific situations which arise intentionally in the SSCI (and Varian) CANDU executive software. Single case violations are handled accurately according to extensive testing that was done, but even that testing was not 100% exhaustive.

In the end, there is reliance on the CANDU DCC executive software to cause a computer restart or program failure on any unintended memory protect violation. And the emulator is assured of handling the intentional violations accurately, at least those cases in Lepreau DCC executive software.

SCOPE AND PRIORITY OF INTERRUPT EMULATION

The only internal interrupts that are emulated are the real time clock interrupt and memory protect violation interrupts. Parity errors, power down, power up, virtual console, and UART interrupts are not emulated.

Restart is emulated (as caused by a watchdog timeout), although it is not an interrupt in the normal sense. When it happens it takes priority over ANYTHING else.

The full external interrupt system is emulated. This includes 3 SSCI PIM's and 4 CAE PIM's providing effectively 52 distinct priority levels of external interrupts.

The real time clock interrupts and external interrupts are grouped together in the way they are handled because both can be held off by uninterruptable machine instructions being executed. These interrupts do not occur while an individual SSCI instruction is in progress of being executed. As each SSCI machine instruction in a computer program concludes being executed, a setting is made in the hardware that either allows or does not allow an interrupt to intervene before the following SSCI machine instruction in the program to be executed. Uninterruptable instructions are those that do not allow an interrupt to intervene before the next SSCI machine instruction gets executed.

When they are allowed to occur, only the highest priority interrupt is acted upon. The real time clock has higher priority than any external interrupt, and the priority of the external interrupts is established by the PIMs hardware.

Memory protect violation interrupts take priority over anything in the emulator, aside from a restart. Memory protect violations only arise as a result of emulating the execution of SSCI machine instructions - but when they arise they are immediately acted upon. So no instruction is uninterruptable with respect to a memory protect violation.

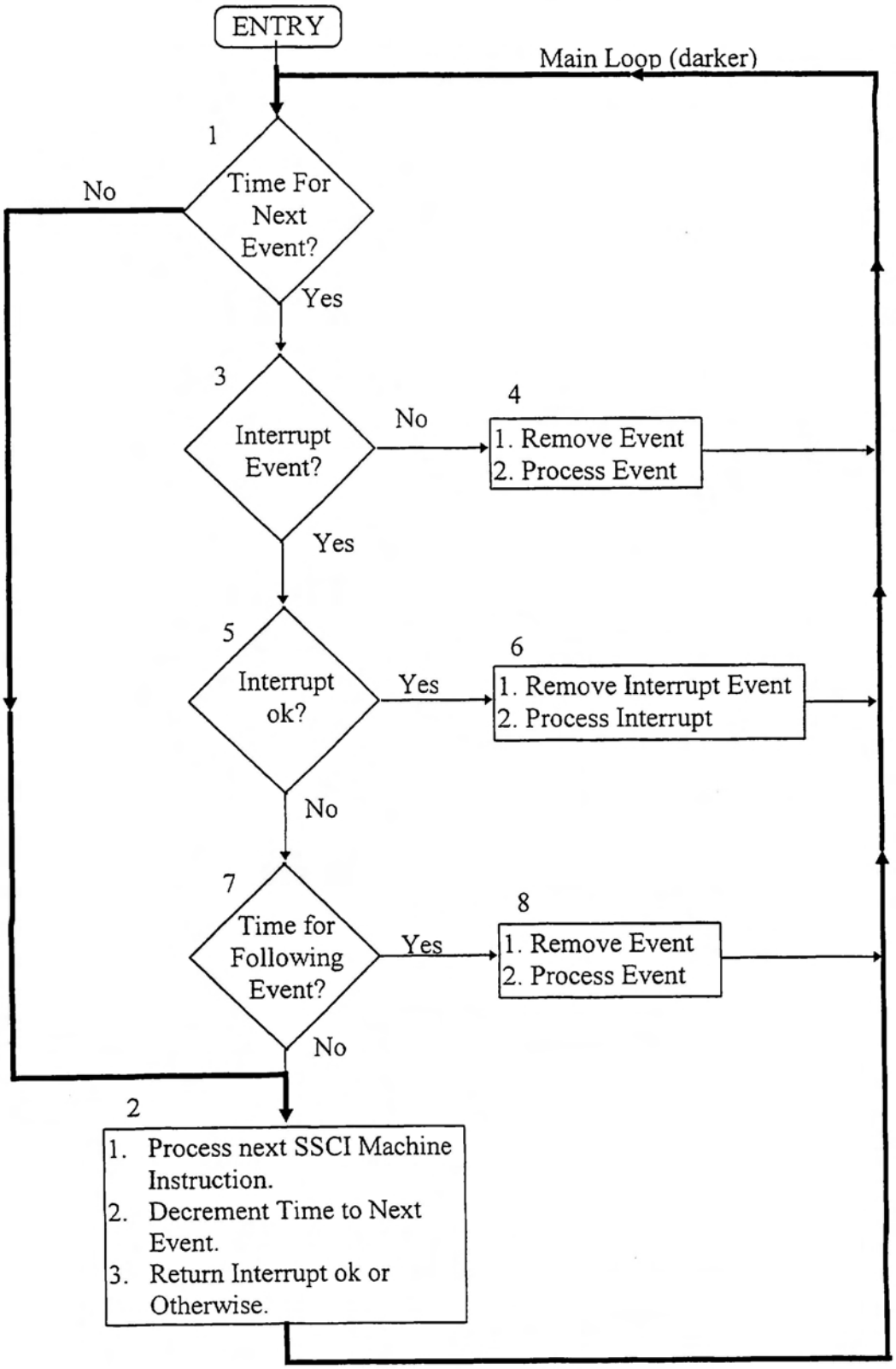
The interruptability characteristics of individual SSCI instructions in respect to clock interrupts and external interrupts is difficult to establish. This information is important for writing device drivers or executive software, and should have been available from documentation but was not. The easiest method to get the information was to disassemble the SSCI microcode for all the SSCI machine instructions. Jump and Mark instructions are uninterruptable if and only if the jump condition is satisfied. If the Jump and Mark is unconditional, the instruction is uninterruptable. Most, but not all I/O instructions are uninterruptable, and the sense instructions are conditionally uninterruptable similar to the jump and marks. Unlike the Varians, double word instructions are generally interruptible on an SSCI, which was a great headache in developing the executive software for the Lepreau simulator DCC because of the different basis for the plant DCC executive software.

THE TOP LEVEL EMULATOR LOGIC

Figure 1 illustrates the top level logic of the emulator. The boxes (diamonds and rectangles) within Figure 1 are numbered 1 to 8 and are each described in turn. The heavier lines going between boxes 1 and 2 indicate that this is "the main loop" where by far the majority of the execution time is spent.

One enters the logic of figure 1 at box 1 after the SSCI emulator is called by the overall simulation dispatcher program which also calls the modeling of the various process systems of the plant such as boilers, heat transport, turbine, etc.).

FIGURE 1 - OVERALL EMULATOR LOGIC



There are a few steps required to make an orderly transition from the plant modeling to the DCC emulator. Interrupts raised by the modeling are handled by calling the functions that lower the voltage of the appropriate lines going into the appropriate interrupt hardware of the emulator. This happens just before one enters Figure 1 at box 1. Another action on entry to the emulator is to set up a special event on the hardware event queue to happen at a future time equal to the time when the emulator is required to return to its calling program. This event has the unique characteristic that the pointer to its event function is a NULL pointer.

Box 1 is simply a test of the time remaining until the scheduled time of occurrence of the first event in the event queue. If this time is zero or negative, then control goes to box 3, but hundreds of times more often than that, control is given to box 2.

Box 2 uses the memory mapping as previously explained to look up the bit pattern for the next SSCI machine instruction. This 16 bit pattern is used as an index into a 64k array of function pointers. To execute the function which emulates any given SSCI instruction one simply executes the function pointed to by the bit pattern of the instruction[4]. That SSCI instruction function has the responsibility to decrement the time to the first event in the event queue by a discrete amount corresponding to the estimated duration of that instruction on the real SSCI hardware. If this instruction is interruptible it is required to return a value indicating interrupts are OK, otherwise return a value indicating otherwise. It was felt that a good compiler ensures the return value comes back in a register and therefore takes minimal overhead in the main loop[5].When control passes to **box 3**, the time to the next event is generally negative, meaning the event is slightly overdue. This is a consequence of moving time forward by discrete amounts in box 2. So if the event queue is updated by removing the event at the head of the queue (in box 4, 6 or 8 below), the time to the next event must be reduced by the amount of time by which the current event is overdue. In that way the schedule is maintained without accumulating slippage.

Box 3 checks to see if the event at the head of the event queue is an interrupt event. There is only one interrupt event allowed in the event queue. It is always inserted at 0 time delay in the queue, which is generally the front of the queue but on occasion may be behind overdue events. The interrupt event is placed in the queue only when all conditions have been satisfied for either a clock interrupt to occur or an external interrupt, except for knowing whether the last SSCI instruction executed was interruptible. The reason for doing this is because of the desire to keep the number of "if" tests in the main loop to just the single test in box 1. Hundreds of instructions are executed just cycling back and forth from box 1 to box 2, before there is generally a requirement to go to box 3.

---

[4] It is impractical to have 64k different functions to cover all possible 16 bit patterns. So one defines a limited set of functions such that each function executes efficiently even allowing for the variations implied by the range of different bit patterns covered by that one function. Invalid bit patterns point to a function that provides appropriate diagnostic information. These pointers are all set up by a utility program which writes a C language source file that appropriately initializes the 64k array of function pointers. In total the Lepreau emulator has a little over 500 distinct C functions, including all those for hardware devices as well as those that emulate machine instructions.

[5] The original design called for the uninterruptable SSCI instruction functions to call the next instruction function from within themselves, instead of calling all instruction functions from within box 2 of the main loop. This leads to a requirement that such instruction functions be reentrant. This might seem problematic but would only have required careful programming for those few SSCI instructions which can directly cause non-memory protect interrupts (e.g. unmasking a priority interrupt module). The reason for finally abandoning this approach was to avoid unjustified delays in the occurrence of hardware events, that are now allowed to happen on schedule thanks to boxes 7 and 8 of figure 1.

The commonest event is the 1 millisecond periodic update of the global time value used by the countdown registers. So it is more common to go from box 3 to box 4, than from box 3 to box 5.

When a countdown register times out, that also causes a branching from box 3 to box 4. The countdown register time out event causes a function call to lower the voltage on a line going into one of the priority interrupt modules (PIMs). Depending on the state of the PIM and the state of the PSW, that may or may not result in an interrupt event being placed on the event queue.

In **box 4** the event data structure is removed from the head of the event queue, and the function which was pointed to by the function pointer in that data structure is executed. In the case of a NULL pointer to the event function, the emulator returns to the dispatcher program that called it.

Box 5 tests the return value from the last SSCI instruction that was executed to determine if it was interruptible or not. Usually the previous instruction is interruptible, in which case control is transferred to box 6.

In **box 6** the event data structure is removed from the head of the event queue, and the interrupt function which was pointed to by the function pointer in that data structure is executed. The interrupt function checks interrupts top down in priority order until it finds an interrupt that is ready to go. This involves checking the real time clock first[6] and then the state of the Varian PIMs. Only the highest priority interrupt is a concern because the hardware immediately disables all interrupts as soon as the top priority one gets through.

When any SSCI instruction is subsequently executed that may make it possible for an interrupt to come in, then there is a possibility that an interrupt event may again be inserted on the event queue depending on the status of the PIMs and PSW.

When an SSCI instruction is executed at any time that has the potential effect of blocking an interrupt from coming in, there is no attempt to remove an interrupt event already on the event queue, because an assessment of the interrupt system would be needed before it could be known if it were correct to remove the event. Instead the interrupt event is simply allowed to occur. The interrupt event function checks interrupts in top down priority and therefore ends up checking all interrupts, if no interrupt is there. The interrupt event is removed in any case, and it has no effect if there is no interrupt set to go.

Box 7 is for those cases where there is an interrupt at the head of the event queue, but it is being held off by at least one (and perhaps several consecutive) uninterruptable SSCI instructions that are handled in box 2. While the interrupt is stuck at the front of the queue, the logic is required to look behind the interrupt event, to see if the following event is due to happen. By design there can be only one interrupt event, and so the next event is definitely not an interrupt. If the next event is due, control is transferred to box 8.

In **box 8**, the non-interrupt event that is due has its data structure removed from the event queue, and the function is executed which was pointed to from the removed event. In the case of a NULL function pointer, the emulator returns to the simulator dispatcher program which called it.

USER INTERFACE

There is a mimic of the DCC keyboard in an alpha display window such that a mouse "operates" the keys on the DCC keyboard mimic.

---

[6] The real time clock interrupts periodically by having an event that reschedules itself according to the clock period set in the hardware (a 4 milliseconds period is set up in the SSCI executive).

RAMTEK channel 2 is emulated in another alpha display window. This is the reactor regulating system RAMTEK DCC display channel corresponding to the emulated keyboard, showing the results of keyboard actions.

Because of the design of the DCC software it is easy to call any function on any DCC keyboard from this one keyboard/display pair. It is also easy to dump a copy of any portion of the alpha screen to a printer.

RAMTEK Channel 0 is emulated in another alpha window. This is the first annunciation display channel in the training simulator control room.

The emulator includes DCC printer capabilities where the printer output goes to a "window" on the alpha, which one can save as a text file for analysis or which can be printed out.

With the 17" display monitor it is normal to have all four of the above windows effectively visible at the same time: keyboard, 2 RAMTEK display channels, and printer output.

Another window is for the software debugger which can be called up on the alpha as required.

In addition to the foregoing, a complete desktop simulator also includes an instructor facility PC exactly the same as is used in the training simulator control room by the simulator instructors to run their lesson plans, record the results of simulations, control the insertion of thousands of different malfunctions, and generally manage the training experience. The same instructor facility software is used on the desktop simulator in conjunction with the DCC emulator.

EXTENT OF DEVICE EMULATION

Instructions covered by the emulator include everything necessary to run the original SSCI executive software for the Lepreau training simulator, and perhaps more. This instruction set covers the V70 series of machines pretty well. The format 18 byte oriented instructions of the SSCI are not emulated as these are not used in our software: the SSCI is basically a clone of the 16 bit word oriented Varians. The format 26 double word move instructions are not emulated, nor are the Format 20 double precision instructions. The format 3 instructions (branch to control store, interpreter decoder) are not emulated and format 21 is apparently not used in the SSCI itself. Without exhaustive checking, it is at least close to the truth to claim that all instructions in all other formats have been emulated.

The I/O instructions include operations on an AI IOBIC, a display IOBIC and DIC, a BIOC subsystem (DI's, DO's, AO's, watchdog timer, CDR's and CAE PIM mask registers), two BMU controllers and associated BIC's, a STATOS controller and an associated BIC, and a VIC plus the previously mentioned SSCI PIMs and CAE PIMs.

The training simulator includes a Dacbus - Unibus Smart Controller (DUSC) providing interfacing between the SSCI and the VAX 4105 modeling computer of the training simulator. The DUSC is imitated functionally, rather than doing a hardware emulation of its instruction set.

Functions performed by the DUSC include a sampling of the SSCI timed DO's to provide the modeling with information on what fraction of time each timed DO has been closed over a recent 200 millisecond period. A similar service is provided in the SSCI emulator with an ancillary process for the timed DO's where the function conducting this process is run from an event on the hardware event queue of the emulator. The event reschedules itself every 10 milliseconds, which corresponds to the sampling rate used by the DUSC for looking at the timed DO's.

Another DUSC function is the feeding of contact alarm interrupts into the SSCI from a buffer of such alarms computed by modeling. A functionally similar process is set up in the emulator.

The video hardcopy controller associated with the SSCI printer is not emulated because we have a built in graphics screen dump capability in the alpha windowing software. The gateway PDLC and associated BIC are also not emulated, as the gateway data dumping software is not usually in service on the training simulator.

VALIDATION OF THE EMULATOR

The emulator is not fully validated at the time of writing this document because there is not yet a sufficient body of experience using it.

When the abstract was submitted for this paper about 3 months ago the only programs that had been run on the emulator were limited scope test programs. Significant sections of the emulator were not yet written, and no integrated testing had been done.

During earlier development a concerted effort was made to develop test programs for all the interior (i.e. non- I/O) SSCI machine instructions where these test programs self checked for certain results. The test programs were checked by running them on the real SSCI hardware, and then used to check the emulator by running the test programs on the emulator.

As all the pieces of the emulator came together, the emulator was asked to run the SSCI executive software. Programming and emulator specification errors were discovered from this, and corrected.

Once the SSCI executive ran on the emulator, the emulator was integrated with the full scope modeling of the Point Lepreau Nuclear power plant, and integrated testing continued. This resulted in the discovery and correction of some problems in the interfacing between the emulator and the modeling. About two weeks ago the DCC software running on the emulator first succeeded in keeping the modeled power plant at full power steady state.

Since then debugging has continued, primarily on the RAMTEK display system emulation. Several overall plant transients have been run and examined with excellent results. Details of comparisons between the training simulator and the desktop simulator are given in the companion paper (1).

Successful execution of numerous complex overall plant transients in a manner that duplicates the performance of the training simulator is highly persuasive of the validity of the emulator.

By the time this paper is presented in 7 weeks from now, it is anticipated there will be a lot more to say about experience using the emulator in day to day work at Point Lepreau.

ACKNOWLEDGMENTS

Evan Young of the Core Monitoring group at Point Lepreau was perhaps to first to remark on the value of DCC emulation to non-real time simulation applications. This became a major consideration in emulator design.

The project never would have happened had it not been for the management support of Joe McCarthy and Paul Thompson. The encouragement of Syd Turner in the early days of conceiving the project is also much appreciated.

Almost all the coding was done by undergraduate University of New Brunswick "co-op" computer science students. No NB Power person was dedicated full time to this project. Brian Benwell, the first of these students to work on the project, made a major contribution in the development of the specification based on many hours of testing the SSCI hardware, assembling, evaluating and annotating manufacturer's documentation.

Actual coding took place over a 16 month period starting in January 1996 principally by two students, one following the other, each on an 8 month work assignment. The first of the 2 main coders was Michelle Perry who wrote and tested the first versions of most of the functions for performing non-I/O instructions. She was followed by Philippe Heroux who had the job of writing the hard-to-test I/O instructions plus the fun of going back and fixing all the things we discovered were needing refinement in our original specification, and then the fun of putting everything together and making it work. Philippe made contributions refining the way things were done throughout the emulator - his role was crucial[7]. Other students on shorter 4 month work terms also made some contributions to the emulator based on a portion of the shorter time they were with us: these were Steve Miller and Devin Manes. The clarity and quality of the code in the emulator is well beyond what one might expect for a group of undergraduate students. The code is easy to read and understand.

REFERENCES

Refer to the following companion paper for a more complete description of the desktop simulator of which the Point Lepreau DCC emulator is a component.

(1) M.MacLean, et al, New Brunswick Power, "The Point Lepreau Desktop Simulator", 18th Canadian Nuclear Society Annual Conference, Toronto, June 8-11, 1997.

Previous papers on DCC emulators known to the authors are as follows.

(2) Dan Hamden, Simulator Services Dept. of Ontario Hydro, "Varian V72 DCC Realtime Software Emulator", CANDU Owners' Group 2nd CANDU Computer Conference, Toronto, Oct 1-3, 1995

(3) A.M. McDonald, et al , Atomic Energy of Canada Ltd., "Development of DCC Software Dynamic Test Facility - Past and Future", 17th Canadian Nuclear Society Annual Conference, Fredericton, New Brunswick, June 9-12 1996.

[7] Remarkably Philippe was on his very first work assignment as a co-op student and had his 20th birthday during his 8 month stay with us.