An Interrupt Timing Simulation

V. D. Jones and R. L. Schwartz
DSN Data S.items Development Section

This article describes a timing simulator written in ANSI FORTRAN. The program was developed to aid in the location of timing anomalies in existing interrupt-driven software and to assist in the design of new real-time programs.

I. Introduction

The timing simulation is designed to simulate the real-time processing of internal and external interrupts of both an asynchronous and synchronous nature as well as a series of subprograms that are to be executed as a result of executing an interrupt processing routine.

In order to make use of the simulation, information must be known as to the exact structure and hierarchy of the processing system. The relative priority as well as the duration and frequency of the different routines must also be known.

The simulation handles virtually any system consisting of a supervisory loop with asynchronous and synchronous interrupts and a subprogram queue structure. When all information concerning the system has been given to the program, the simulation of the interrupt handling will occur. Various statistics and state snapshots are available.

A brief timing study of the effect of a 20-character per second Remote Output Teletype on the Viking Telemetry and Command Processor (TCP) using the XDS 920 computer is provided as an example.

II. Overview

The simulator accepts as input data mnemonics which represent Primary Interrupts (PINs), Clock Interrupts (CLKs), and subprograms in a priority queue (PSQ). Priorities, durations, and interval times (not for all inputs) are given (in number of cycles) and the handling of the interrupts is simulated.

At the beginning of each cycle, every clock is checked to see if its interval has expired.\(^1\) If so, the clock is placed on a push down stack. The primary interrupts are then checked to see if there are any occurrences for that cycle, and, if so, the interrupt is put in the stack. The stack is then sorted according to the priority of each interrupt. At

\(^1\)In actuality, a computation is done to determine the next cycle at which a change occurs, and all necessary counters are incremented to that value, eliminating the execution of needless cycles.
this point, the duration counter of the highest priority interrupt is updated. If this is the first cycle of the routine, then the elements in the PSQ associated with that interrupt are enabled. When the duration is satisfied, the interrupt is wiped off the top of the stack. If there are no interrupts to process during that cycle, then the highest priority subprogram that is enabled is fetched and put in the stack. When no PSQ entry has been enabled, the program loops in the supervisor (MAIN) waiting for an interrupt to occur.

Additional features include the ability for a PSQ element to queue other PSQ elements or to enable clock interrupts, and for a clock to have a controlling envelope.

A watchdog timer interval may be specified in order to monitor the activity of the program. This gives status of all interrupts and subprograms. Following the final time specified for the simulation, various statistics are given concerning the activity of the interrupts during the run.

III. Structural Elements of Simulation

A. Priority Subprogram Queue (PSQ)

The PSQ routines are those which are executed in the base or noninterrupt status of the machine. They exist in a priority queue and are enabled by clocks, primary interrupts, and other PSQ routines. Each PSQ can either be assigned a priority as with the clocks and primary interrupts, or will be assigned a default priority of zero, with its position in the PSQ table (relative to the top) determining its priority with respect to other PSQs. Each PSQ entry is disabled after execution. The “main loop” routine is a special case of the PSQ type. It is always enabled and has the lowest priority, thus it will only be executed when nothing else is active or pending.

B. Clock (CLK)

The clock routines are periodically enabled as by a cyclic interrupt. They are given priorities which determine their relative importance with respect to the other clocks, primary interrupts, and, in some cases, the PSQs. The higher the number, the higher the priority. Once started, a clock is enabled once each time its interval expires. This is subject to conditions specified below in regard to the envelope and clock flags. A clock may have a random or deterministic start time. For a random start time, a random number between 1 and 255 is assigned. The duration of the clock interrupt handling routine is also given. When a clock has been enabled and reaches the top of the stack, it is executed. On the first cycle of execution each CLK enables the PSQs associated with it.

C. Primary Interrupt (PIN)

The PIN routines are asynchronous interrupts whose enabling times are given in the input data. When the PIN occurs, it is enabled, and the first cycle of its operation is used to enable all PSQs associated with it. Since it is asynchronous, no envelope control is required. Each PIN has a priority and duration.

D. Envelope (ENV)

Envelopes or ENVs serve to inhibit or permit clock interrupts. If a clock interrupt is controlled by an envelope, an interrupt is only recognized when the envelope is high. A clock can be controlled by at most one envelope, but an envelope may control more than one clock. The waveform of the envelope is similar to that of the clocks. It is started at a time either specified as random or deterministic. Each ENV is given a duration and an interval. Thus, if an envelope is turned on at time $t_o$, it will be on until $t_o + t_d$. It will be off from $t_o + t_d$ until $t_o + t_i$, then on from $t_o + t_i$ until $t_o + t_i + t_o$, etc.

E. Clock Control Flag (CCF)

Clock control flags or CCFs are controlled by the completion of PSQs. CCFs exist as a simulation feature to allow disabling and enabling of interrupts by subprograms. Clocks may be controlled by more than one PSQ, and a PSQ may control several clocks.

F. ENV–CCF Interaction

The envelopes (ENVs) and the clock control flags (CCFs) are examined, and, if “true,” the appropriate clocks will be enabled. If a clock is disabled when it would normally start as the result of interval expiration, it will not start until the interval has expired (again).

IV. Modified Backus Normal Form (BNF)

Description of Simulation Input Data

Input editing is structured to process the mnemonics and associated parameters in accordance with the syntax rules given below. The notation is to be interpreted as follows:

1. Read the connective “:=” as “is formed from”
(2) Read the symbol "<X>" as: "the object named X" shortens the sequence given by:

<\mathbf{X} > := p \\
<\mathbf{X} > := q \\
<\mathbf{X} > := r \\
(3) \{ \cdot \}_a^b \text{ indicates repetition of the set } \{ \cdot \} \text{ from } a \text{ to } b \text{ inclusive} \\
(4) \{ \cdot \}_a^a \text{ indicates repetition of the set } \{ \cdot \} \text{ a times} \\
(5) "|" \text{ indicates exclusive or; viz. } <\mathbf{X} > := p | q | r \\
(6) \text{ Any string not enclosed in angular brackets } (<\cdots>) \text{ represent a terminal element, e.g., } +, -, \text{ RND, } \& (\text{blank}), \$, etc.

\begin{align*}
\langle \text{DIGIT} \rangle & : = 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 \\
\langle \text{CHAR} \rangle & : = A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z \\
\langle \text{DELIMITER} \rangle & : = \ldots | \text{ } \\
\langle \text{TERMINATOR} \rangle & : = \$ \langle \text{NOISE} \rangle \\
\langle \text{NAME} \rangle & : = \langle \text{CHAR} \rangle \{ \langle \text{CHAR} \rangle | \langle \text{DIGIT} \rangle \}_0^5 \langle \text{DELIMITER} \rangle \\
\langle \text{NUMBER} \rangle & : = \{ \langle \text{DIGIT} \rangle \}_1^8 \langle \text{DELIMITER} \rangle \text{ (UNLESS NOTED ELSEWHERE) } \\
\langle \text{NOISE} \rangle & : = \langle \text{NAME} \rangle | \{ \langle \text{NAME} \rangle \langle \text{DELIMITER} \rangle \}_0^\infty \\
\langle \text{NUMERIC DATUM} \rangle & : = \langle \text{NOISE} \rangle \langle \text{NUMBER} \rangle \langle \text{NOISE} \rangle \\
\langle \text{START TIME} \rangle & : = \langle \text{NUMBER} \rangle \text{ RND} \langle \text{NUMBER} \ (\leq 255) \rangle \text{ RND} \\
\langle \text{FLAG} \rangle & : = + | - \\
\langle \text{TFIN} \rangle & : = \langle \text{NUMERIC DATUM} \rangle \\
\langle \text{PSQ} \rangle & : = \langle \text{NAME} \rangle \langle \text{NUMBER (DURATION)} \rangle \langle \text{NUMBER (PRIORITY)} \rangle \{ \langle \text{NUMBER (NUMBER OF PSQ\_NAME)} \rangle \} \{ \text{NUMBER PSQs} \}_1 \{ \text{NUMBER PSQs} \}_0 \\
\text{NOTE: A PSQ may call only those PSQs already input.} \\
\langle \text{MAIN LOOP} \rangle & : = \langle \text{NUMERIC DATUM} \rangle \\
\langle \text{WATCHDOG} \rangle & : = \langle \text{NUMERIC DATUM} \rangle \\
\langle \text{CLOCK} \rangle & : = \langle \text{NAME} \rangle \langle \text{NUMBER (PRIORITY)} \rangle \langle \text{NUMBER (INTERVAL)} \rangle \langle \text{NUMBER (DURATION)} \rangle \langle \text{NUMBER (NUMBER PSQ\_NAME) ENABLED)} \rangle \langle \text{START TIME} \rangle \{ \langle \text{NAME (PSQ\_NAME)} \rangle \} \text{NUMBER PSQs} \\
\{ + \langle \text{NUMBER (NUMBER PSQ\_NAME) ENABLED) } \rangle \\
\{ \langle \text{NAME (PSQ\_NAME)} \rangle \} \text{NUMBER PSQs} \}_n \{ \langle \text{NAME (PSQ\_NAME)} \rangle \} \text{NUMBER PSQs} \}_0 \\
\text{NOTE: "NUMBER OF PSQ\_NAME ENABLED" reflects the number on that card. Continuation lines contain a separate count. } 0 \leq n = \text{total number PSQs enabled} \leq 21. 
\end{align*}
V. Viking Telemetry and Command Processor Timing Simulation

After determining the program structure, as well as the execution time and frequency (where applicable) of the routines, the input data (Fig. 1) reflect the simulation input with worst case timing estimates. Figure 2 illustrates a snapshot of the simulated system following expiration of total process time.

These results apply to the Viking Model with the following current operations:

(1) 20 characters per second teletype outputting continuous alarms
(2) 33⅓-bit per second Channel 1 telemetry.
(3) 2000-bit per second Channel 2 telemetry.
(4) 16000-bit per second Channel 3 telemetry.
(5) 4-bit per second command rate.
(6) 11-word Mission Definition Table transfer from TCP to DDA (Data Decoder Assembly) every second.

The above system was run for 4 seconds (500,000 cycles) with telemetry acquisition lasting 1.26 seconds (157,584 cycles) and tracking for 3.74 seconds (342,416 cycles).

The worst case analysis yielded the following: During Channel 1 acquisition an overrun of the module status subprogram (MSTAT) queued by the 10-pulse per second interrupt routine occurred 67% of the time. This was due to both the increased execution time of the
Channel 1 trap (CITRPA) which takes roughly twice as long during acquisition and to the processing of GCF blocks and the message processor done by the GCFBLK and MSGPRC subprograms. Despite the MSTAT overrun, status was still completed approximately every 315 milliseconds.

During tracking the overrun of MSTAT was less severe, with an overrun only occurring 25% of the time yielding a completed module status update approximately every 180 milliseconds.

An overrun of MSTAT was not caused to any large extent by the addition of the 20-cps teletype but was caused by the increased activity of the Channel 1 trap during acquisition and the long execution time for MSGPRC and GCFBLK (approximately 120 milliseconds).

If the module status is desired more frequently than every 315 milliseconds during acquisition and every 180 milliseconds during tracking, it is suggested that the MSGPRC and GCFBLK subprograms be segmented into four smaller blocks to allow the executive to check for enabled subprograms of higher priority between segment execution.

VI. Program Usage

The program (written in ANSI FORTRAN) takes approximately 14K of memory (7K for instruction bank, 7K for data bank). See the Timing Simulation Descriptive Document (Ref. 1) for a complete description including operating instructions for the UNIVAC 1108 and the XEROX SIGMA 5, and a description of how to construct a simulation given an interrupt processing system.

Reference


Bibliography


Fig. 1. Sample input data for Viking telemetry and command simulation
Fig. 2. Sample output of Viking telemetry and command model