NASA Technical Memorandum 100478

IN-CIRCUIT FAULT INJECTOR USER'S GUIDE

Peter A. Padilla

JUNE 1987

(NASA-TM-100478) IN-CIRCUIT FAULT INJECTOR USER'S GUIDE (NASA) 24 p Avail: NTIS HC A02/MF A01 CSCL 09C

N87-27910

Unclas
G3/33 0093217

National Aeronautics and Space Administration

Langley Research Center
Hampton, Virginia 23665

SUMMARY

Fault injection experiments are performed as part of validation research of fault tolerant computer systems. The fault injection experiment data provide information on the system's hardware fault coverage, and on the overall fault handling behavior of the total system. This information can be used for reliability analysis and system modeling of the fault tolerant computer system.

A fault injector system has been designed and developed to facilitate fault injection experiments. The fault injector system, called In-Circuit Fault Injector (ICFI), allows fault injections to be performed on electronic systems without special test fixtures. The fault injector probes are attached directly to dip clips which can be attached to the integrated circuits on the system boards. Once connected, the ICFI system can be used to inject stuck-at-one, stuck-at-zero, square wave and transient faults. For stuck-at faults the injection time can be specified to be from 500 nanoseconds to 10.9 minutes. Faults injected on buffers, line drivers, and/or bus transceivers must be limited to 30 seconds to assure the in-circuit fault injection technique will not damage or destroy the device under test.

The ICFI system is interfaced to a VAX-11/750 minicomputer. Software has been developed in the VAX minicomputer to control the ICFI system remotely. This interface allows researchers to execute un-attended fault injection experiments.

The control software, a device driver, was developed in VAX assembly language. The device driver is an integral part of the VAX operating system and can be accessed through system services. Detailed definitions of the parameters and examples of the computer code required to access the device driver are presented.

Also presented, is the connection procedure to be followed to connect the ICFI system to a circuit and the front panel controls which allow manual control of fault injections.

INTRODUCTION

This document is a comprehensive user's guide for a fault injector system used in the Avionics Integration Research Lab. (AIRLAB) for fault injection experiments on a fault tolerant computer system test-bed.

Fault injection experiments are performed as part of validation research of fault tolerant computer systems. The fault injection experiment data provide information on the system's hardware fault coverage, and on the overall fault handling behavior of the total system. This information can be used for reliability analysis and system modeling of the fault tolerant computer system.

Past fault injector systems required boards with sockets so that the integrated circuits (ICs) could be removed, and the fault injector connected between the ICs and the board. This technique also requires special implants that could be inserted on the board's sockets and that provide a socket for the displaced IC. Drawbacks of this technique include:

A) high cost of building boards with sockets

B) implants add capacitance and inductance loading to the circuit under test

C) implants have to be carefully designed and layed-out to minimize cross-talk and high frequency feedthrough

D) cumbersome set-up procedure:

1. remove target board

2. remove target chip

3. connect implant to board

4. connect chip to implant

5. put board with implant back into the system

E) size of the implant makes it difficult to hold it in place.

The fault injector system described here, called In-Circuit Fault Injector (ICFI), provides the capability of injecting faults in digital circuits without the use of special test fixtures, e.g., board sockets. The ICFI system is connected to the target ICs through dip clips. This connection minimizes capacitive loading while avoiding inductive loading. The connection also avoids cross-talk or feedthrough by not being in series with the system's signal path. During a fault injection, the ICFI hardware pulls the system-under-test circuit nodes to a specified voltage level. The ICFI hardware and control/interface software provide an integrated system for the development of application programs to automatically execute fault injections.

There are certain precautions that must be taken with the in-circuit fault injection technique which, if not taken, may damage or destroy a chip. Specifically, the fault voltage should not be set outside the target device's output voltage range, e.g. the fault voltage should be limited between +5 to 0 volts for TTL devices. Another precaution that must be taken with this technique is the fault duration limit that exists for line drivers, buffers, and bus transceivers. Faults injected at the output of these devices must be limited in duration to 30 seconds.

A VAX/VMS device driver to control the ICFI hardware was developed. Also, a user interface to the device driver, written in PASCAL, was developed and is described in reference 4. This interface provides a user friendly access to all the ICFI system capabilities and satisfies the requirements for most experiments. Experimenters with special needs, not addressed by the interface program, will have to develop their own user interface to the device driver.

This paper describes the device driver programming interface and the ICFI hardware set-up necessary for automatic fault injections.

DEVICE DRIVER: USER-PROGRAMMING INTERFACE

The ICFI device driver was written in VAX macro and is installed at system start-up. The device driver is an integral part of the VAX operating system (in this case VMS) and can be accessed through VMS system services. The system services can be accessed through any high level language or from VAX macro assembly. VAX macro assembly was selected as the language for the examples presented in this section.

It is not the intention of this paper to teach VAX macro assembly language but to provide a guide on how to write application programs for the ICFI system. Examples are given on the required code to do this; the examples are syntactically correct and familiarity with VAX macro is assumed. For detailed information on VAX macro, system services, and device drivers consult Digital Equipment Corp. manuals, refs. 1, 2 & 3.

The first system service required to gain access to the ICFI device driver is the $ASSIGN system service. This service requires the user to supply two parameters, the name of a device to which an I/O channel is desired and a location to receive the allocated channel number. In general, the macro code required is:

```
device:  .ascid /device_name/
channel: .word 0

         .entry example
         $assign_s device,channel
```

The first line in the example above creates a descriptor, named "device", that contains the device name. The second line reserves a memory location named "channel" and initializes it to zero. The third line creates an entry point to the executable program named "example" (where it starts execution every time you call the program with the DCL command RUN). The fourth and last line in the example is the assign system service which allocates an I/O channel to the device specified in the first line in the example and stores the channel number in the memory location identified by the "channel" label. The "underscore" and the "s" attached to the assign keyword are required (they indicate to the macro assembler that the stack should be used to pass parameters).

There are three devices which are the front end (from the VAX side) of the ICFI system. These devices are:

device physical name	device function
AOA0:	D/A converters
ISA0:	Digital outputs
KWA0:	16-bit timer

Using the example above and the ICFI device names we can develop an example of the code to allocate an I/O channel to each device:

```
device_1:  .ascid /AOA0:/
device_2:  .ascid /ISA0:/
device_3:  .ascid /KWA0:/
channel_1: .word 0
channel_2: .word 0
channel_3: .word 0

        .entry example,^M<r2,r3,r4,r5>
        $assign_s device_1,channel_1
        $assign_s device_2,channel_2
        $assign_s device_3,channel_3
```

The ^M<r2,r3,r4,r5> in the entry statement tells the program caller (usually the operating system) to push r2,r3,r4,r5 on the stack when calling the program (the caller must use the "CALL" assembly instruction) and to pop them after the end of the program (by "RET" assembly instruction). Refer to .ENTRY and .END macros in reference 1 and $EXIT system service in reference 2. The registers specified inside the brackets should be those used in the program except for r0, r1, r12, or r13. This example is syntactically correct.

After obtaining the I/O channels to these devices a second system service (QIO system service) is required to tell the device driver what to do. This system service is used several times to pass data to the driver. The driver will use this data to setup the fault injector for an experiment. An example of the code required for a QIO call to set-up the ICFI system is:

```
efn = 0
io$_da = ^x001E

        $qio_s efn,channel,#io$_da,,,,P1,P2,P3,P4,P5
```

or

```
$qiow_s efn,channel,#io$_da,,,,P1,P2,P3,P4,P5
```

where "efn" is a flag to be set after the QIO call completion (usually set to zero); "channel" has been explained in the $assign system service example; "io$_da" is a function number used by the driver to determine what to do (explained in more detail later); and "P1" to "P5" are parameters (user specified data). The "^x" in the "io$_da" assignment statement means the number that follows is in hexadecimal format. The "w" in the second QIO means that the system service will suspend the program's execution until the QIO is finished. In the first QIO, the process and the system continue execution asynchronously. For details on the QIO system service refer to ref. 2.

As mentioned above there are three front end devices in the ICFI system, which can be accessed using the I/O channels previously allocated. Each device has different functions to perform and the device driver (which contains the control software for each device) most be told what function the user wants performed. To do this, the user must pass the function code (e.g., "io$_da" in the QIO call above) and the channel number to the driver, see figure 1.

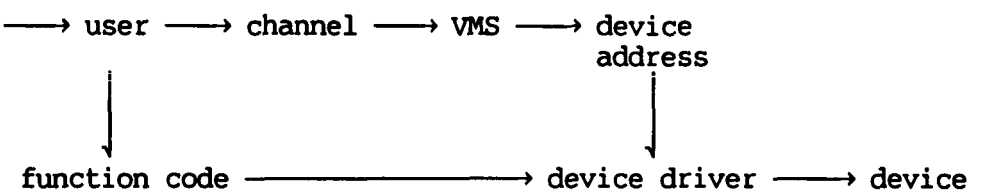

Figure 1. Information flow-graph for correct user-to-device communications.

The channel used in the QIO call determines the device to which the device driver will talk. The function code determines the parts of the device driver code which will be executed. Disagreement between the function code and the channel, i.e., using a function code which specifies a function which is not performed by the device specified by the channel, will CRASH THE SYSTEM. Table 1 specifies the valid function codes for the three front end ICFI system devices.

TABLE 1. – VALID FUNCTION CODES FOR ICFI FRONT END DEVICES

device	function code	function
AOA0:	$001E_{16}$	Set D/A's
ISA0:	$000D_{16}$	Define output probe
	0004_{16}	Reset
	$000A_{16}$	Fault type and set hardware timers
KWA0:	0006_{16}	Fault type and set KWA0: timer

The single valid function code ($001E_{16}$) for the D/A device "AOA0:", shown in table 1, tells the device driver to execute the code that checks the validity of the P1, P2, and P3 parameters, shown in table 2. (NOTE: the P1, P2, P3, P4, and P5 parameters for all the function codes are defined in tables 2, 3a, 3b, and 4 with detailed explanations following in the text.) Using these data the device driver sets the D/A's to the appropriate output voltage level.

The first function code ($000D_{16}$) for the digital outputs device "ISA0:", shown in table 1, enables one or two ICFI system output probes (specified by P1 and P2 in table 3a) where a fault injection is desired. The second function code (0004_{16}) forces the device driver to reset the ICFI system. The third, and last, valid function code ($000A_{16}$) for this device defines ,through P1 and P2 (table 3b), the fault type (e.g., stuck-at-zero, stuck-at-one, and others defined below) for the two output probes enabled, loads three hardware timers with user defined values passed in P3, P4, and P5 (table 3b), and starts the fault injection.

The function code (0006_{16}) for the 16-bit timer device "KWA0:" also defines the fault type for the two previously enabled output probes (table 4). But, instead of the ICFI system hardware timers, it uses a separate 16-bit timer which is loaded with data passed on P3 and P4 (table 4). This function code starts a fault injection.

As mentioned above, the QIO parameters P1 to P5 are user defined, within the restrictions defined and explained in more detail below for each table 2 through 4.

TABLE 2. - QIO "P" PARAMETER DEFINITIONS FOR THE D/A DEVICE

function code (device) function	parameter definitions
$001E_{16}$ (AOA0:) Set D/A device	P1 = V_+ , logic one voltage level (volts) ($-15.0 \leq V_+ \leq +15.0$) Floating point data P2 = V_- , logic zero voltage level (volts) ($-15.0 \leq V_- \leq +15.0$) Floating point data P3 = 16 words buffer address (allocated by user, for driver use only)

For the "set D/A device" function, V_+ defines the voltage level used when injecting a SA1 fault (see also fig. 2). V_- defines the voltage level for the SA0. Care must be taken when defining these voltage levels, they should be within the allowed voltage range of the technology used in the circuit under test (CUT), e.g., TTL voltage range: +5 to 0 volts. (The user interface program described in reference 4 limits these voltages to the TTL voltage range.)

TABLE 3A. - QIO "P" PARAMETER DEFINITIONS FOR THE DIGITAL OUTPUTS DEVICE

function code (device) function	parameter definitions
$000D_{16}$ (ISA0:) Define output probe	P1 = output probe number ($1 \leq P1 \leq 48$) P1 = 0 (no output, disable output) Integer data format P2 = output probe number ($1 \leq P2 \leq 48$) P2 is used for double fault injections only P2 = 0 (no output, disable output) P2 must be zero when injecting single faults Integer data format
0004_{16} (ISA0:) Reset	No parameters needed

For the "define output probe" function, the number of output probes is between 1 and 48, inclusive. The ICFI system output probes are divided in two banks of 24 probes each, i.e., $1 \leq$ bank #1 ≤ 24 , $25 \leq$ bank #2 ≤ 48. For double faults, assigning P1 to bank #1 and P2 to bank #2 allows the selection of different fault types for P1 and P2, e.g., a SA1 could be injected at the output probe defined in P1 and a SA0 at P2. When the first fault type (i.e., for P1) is a stuck-at, the selection for the second fault type (for P2) can be a stuck-at ,transient, or square wave. When the first fault type selected is a transient or a square wave, the second fault type selection is restricted to a stuck-at or to a transient/square wave identical to the first fault type selection. If both P1 and P2 are defined within the same output probes bank number only one fault type can be defined for both, e.g., a SA1 at both P1 and P2.

TABLE 3B. - QIO "P" PARAMETER DEFINITIONS FOR THE DIGITAL OUTPUTS DEVICE

function code (device) function	parameter definitions
$000A_{16}$ (ISA0:) Fault type/hardware timers	P1 = fault type number fault type number: 1 - clear/no fault 2 - stuck-at-one (SA1) 3 - stuck-at-zero (SA0) 4 - transient 5 - square wave Integer data format P2 = fault type number (same as for P1) P2 = 1 , for single fault injections Integer data format P3 = value, in seconds, to be loaded in T1 (see fig. 2 below) 300E-9 $\leq$ P3 $\leq$ 12.8E-6 or P3 = 0 (disable T1) Floating point data P4 = value, in seconds, to be loaded in T2 (see fig. 2 below) 300E-9 $\leq$ P4 $\leq$ 12.8E-6 or P4 = 0 (disable T2) Floating point data P5 = value, in seconds, to be loaded in T3 (see fig. 2 below) 300E-9 $\leq$ P5 $\leq$ 3.27E-3 or P5 = 0 (disable T3) Floating point data

The fault types to be injected at the defined output probes are specified in a QIO call with the "fault type/hardware timers" function code. For double fault injections, if both output probe numbers are within one probe bank, P1 defines the fault type that is injected at both probes. If the output probe numbers correspond to different probe banks then P1 defines the fault type that will be injected at the bank #1 output probe and P2 defines the fault type for bank #2 output probe. In this QIO call, P3, P4, and P5

define fault times for fault types 4 (transient) and 5 (square wave). These times are shown in figure 2.

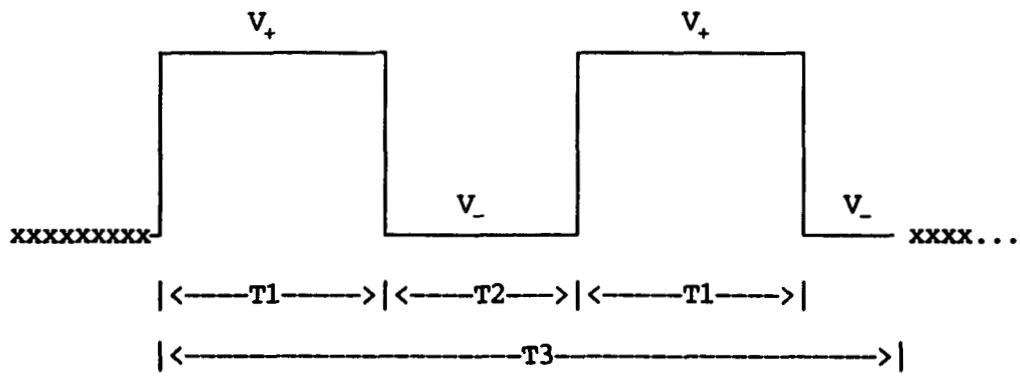

Figure 2. Fault example: definitions of the ICFI hardware timers. The "x" means normal operation of the circuit under test.

For the square wave and transient fault types T1 defines the duration for which V_+ output voltage level is injected. T2 is the duration for which V_- output voltage level is injected for the square wave fault type definition. For the transient fault, T2 defines a duration for which no fault is injected. T3 is the total fault injection duration.

TABLE 4. - QIO "P" PARAMETER DEFINITIONS FOR THE 16-BIT TIMER DEVICE

function code (device) function	parameter definitions
0006_{16} (KWA0:) Fault type/16-bit timer	P1 = fault type number fault type number: 1 - clear/no fault 2 - stuck-at-one (SA1) 3 - stuck-at-zero (SA0) Integer data format P2 = fault type number (same as for P1) P2 = 1 , for single fault injections Integer data format P3 = rate number to be loaded in the KWA CSR Rate number: 2 -> 1MHz 4 -> 100 KHz 6 -> 10 KHz 8 -> 1 KHz 10 -> 100 Hz Integer data format P4 = value to be loaded in the KWA PRESET reg. $0 \leq P4 \leq 65535$ To obtain P4: T_a = fault duration desired (in seconds) R = rate (one of the frequencies above, e.g., 100,000 Hz) $P4 = R(T_a)$ EXAMPLE: R = 1000 Hz T_a = 100E-3 seconds $P4 = R(T_a) = 100$ Integer data format

The "fault type/16-bit timer" function code specifies a fault injection (SA1 or SA0) with a duration controlled by a 16-bit timer ("KWA0:" device). P1 and P2 for this function specify the fault type and are constrained by the same restrictions described for P1 and P2 for the "fault type/hardware timers" function. An additional restriction for these two parameters is the

limited selection of fault types available, i.e., SA1 and SA0 faults only. P3 for this function is a rate number which determines the rate at which the 16-bit timer will count. The rate numbers and the frequencies they specify are given in table 4. P4 specifies the fault duration by the number of clock ticks required at the frequency specified in P3 (see table 4 above for an example).

Having explained all the relevant information we are ready for an example of the code required to inject one fault.

```
device_1:  .ascid /AOA0:/
device_2:  .ascid /ISA0:/
device_3:  .ascid /KWA0:/
channel_1: .word 0
channel_2: .word 0
channel_3: .word 0
io$_da = ^x001E
io$_reset = ^x0004
io$_probe = ^x000D
io$_fault = ^x000A
io$_KW_fault = ^x0006
V+ = ^F5.0
V- = ^F0.0
SA1 = 2
SA0 = 3
buffer: .blkw 16

        .entry example,^M<r2,r3,r4,r5>
        $assign_s device_1,channel_1
        $assign_s device_2,channel_2
        $assign_s device_3,channel_3
        $qiow_s ,channel_2,#io$_reset
        $qiow_s ,channel_1,#io$_da,,,,P1=V+,P2=#V-,P3=#buffer
        $qiow_s ,channel_2,#io$_probe,,,,P1=1,P2=#0

        $qiow_s ,channel_2,#io$_fault,,,,P1=SA1,P2=#1,P3=#^F0.0,-
                P4=#^F0.0,P5=^F0.0
        { add a wait loop here to wait the fault duration }
        $qiow_s ,channel_2,#io$_reset
        $exit_s

or
        $qiow_s efn,channel_3,#io$_KW_fault,,,,P1=SA1,P2=#1,P3=#8,P4=#100
        $qiow_s ,channel_2,#io$_reset
        $exit_s
```

The code shown above assigns channels to the three ICFI system front-end devices. Then, it issues a QIO call to reset the ICFI system. This is just a precaution in case the system is in an unknown state (e.g., if left

injecting a fault by a previous user). The second QIO call sets the fault voltage levels to +5 and 0 volts (TTL levels). The "^F" prefix denotes floating point data. To specify the output probe, where a fault is to be injected (in this case probe #1), a third QIO call with a function code of io$_probe must be issued. After defining the output probe we have two choices for the next QIO call.

The first choice is to inject the fault using the ICFI system internal timers. If the desired fault type is either transient or square wave this is the only choice. If the fault type desired is either a SA1 (like in the example code) or a SA0, this choice restricts the fault duration control to software, i.e., the software has to wait for the duration desired and then clear the fault injection (THE INTERNAL TIMERS T1, T2, AND T3 DO NOT CONTROL THE TIMING FOR SA1 OR SA0 SELECTIONS). To clear a fault a QIO call with the io$_fault function code must be issued after the fault duration time is up (see table 3a).

If the fault duration desired is less than or equal to 3.27 msec. (the maximum value for T3) a SA1 or SA0 fault can be injected with the hardware timers by using the square wave fault type and setting T1 (for SA0) or T2 (for SA1) to zero. The timer not set to zero (either T1 or T2) should be set to its maximum value of 12.8 μsec if the fault duration desired is greater than this, or to the fault duration desired if less than 12.8 μsec. T3 should be set to the fault duration desired. For example if a SA1 fault injection for 2 msec is desired the timers should be set as follows:

T1 = 12.8 μsec
T2 = 0
T3 = 2 msec.

If a SA1 for 10 μsec is desired then the timers' setting should be:

T1 = 10 μsec
T2 = 0
T3 = 10 μsec.

Even while in this option the fault duration is controlled by hardware

timers, the software has to wait for the fault to finish before issuing a QIO call to reset the ICFI system and exiting. If the fault duration for a SA1 or a SA0 is larger than 3.27 msec then the SA1 or SA0 fault type has to be used and the software has to wait for the desired duration before clearing the fault. A better way of controlling the fault duration (for durations longer than 3 msec) for stuck-at faults is the second choice for the fourth QIO call.

The second choice for the fourth QIO call in the example code is to use the 16-bit timer to control the stuck-at fault duration. The timer will interrupt the cpu at the end of the count, thus causing the execution of the ICFI system device driver, which will automatically disable the fault. With the 16-bit timer, fault durations from 1 msec to 655.35 sec can be specified. The lower limit is set by an interrupt latency time of, on average, 40 μsec. This time can not be guaranteed to always be around 40 μsec, it will depend on the number of users and on what is being run on the system. To maintain the error in time to less than 10% a lower limit in the fault duration of 1 msec seems appropriate. More accuracy can be obtained if the desired fault duration is compensated by the 40 μsec average interrupt latency time. The lower limit of 1 msec overlaps with the 3.27 msec limit of the internal hardware timers, thus a continuous fault duration range of 500 nsec to 655 sec is available through internal/external hardware timers.

A QIO call to reset the ICFI system should be issued before exiting a program developed for fault injection experiments. The reset QIO call will reset the ICFI system and put it in a known "disable but ready" state. Currently when a program finishes execution under VMS the device driver is notified with a deassign channel system service call, to which the driver responds by executing code to clear any fault presently enabled in the ICFI system. This prevents users from leaving a fault injection enabled but does not disable the output probes. When fault injections are disabled the lines from the ICFI hardware to the probes are not being actively driven (i.e., passive termination). If the probes are not disabled any noise induced on the lines by Electro-Magnetic Interference (EMI) will cause a undesired fault to be injected. The reset QIO call clears all fault injections and disables all the output probes.

LIMITATIONS OF THE IN-CIRCUIT FAULT INJECTION TECHNIQUE

The ICFI system can perform fault injections on several integrated circuit technologies, e.g. emitter coupled logic (ECL), TTL, complementary metal oxide semiconductor (CMOS), and RS-232 drivers. The voltage range of these technologies are varied, as can be appreciated by the following list:

1. ECL (for the HD10K series, ref. 5)

   a. maximum logic "1" voltage = 0 V
      minimum logic "1" voltage = -1.105 V

   b. minimum logic "0" voltage = -2.0 V
      maximum logic "0" voltage = -1.475 V

2. TTL (ref. 6)

   a. maximum logic "1" voltage = +5.0 V
      minimum logic "1" voltage = +2.0 V

   b. minimum logic "0" voltage = 0 V
      maximum logic "0" voltage = .8 V

3. CMOS (V_{cc} = +5 V, ref. 7)

   a. maximum logic "1" voltage = +5 V
      minimum logic "1" voltage = +3.5 V

   b. minimum logic "0" voltage = 0 V
      maximum logic "0" voltage = +1.5 V

4. CMOS (V_{cc} = +10 V, ref. 7)

   a. maximum logic "1" voltage = +10 V
      minimum logic "1" voltage = +8.0 V

b. minimum logic "0" voltage = 0 V
   maximum logic "0" voltage = +2.0 V

5. RS-232 drivers (ref. 8)

a. maximum logic "1" voltage = +12 V
   minimum logic "1" voltage = +6 V

b. minimum logic "0" voltage = -12 V
   maximum logic "0" voltage = -6 V .

To handle all the technologies mentioned above the ICFI system requires a high output voltage range (±15 volts). This high output voltage range may create problems if precautions are not taken.

The output voltage levels specified in V_+ and V_- in the "set D/A device" QIO call should not be larger than the maximum logic "1" voltage or less than the minimum logic "0" voltage for the particular devices under test. If this precaution is not taken, the devices under test will be damaged or destroyed. The ICFI system allows the users to set V_+ and V_- to any value within the output range. It is the user's responsibility to set these voltages correctly.

Another area of concern is the fault duration when injecting on devices with high output current capabilities like line drivers, buffers, and bus transceivers. The ICFI system injects faults by forcing a circuit node to the output voltage selected. The device-under-test's output driving this node will be subjected to a high magnitude current, the node forced response current. This current increases the device's power dissipation. The high power dissipation, if maintained over certain time interval, will increase the device's temperature to levels which will destroy the device. To prevent this, it is necessary to limit fault injections on these devices to a maximum of 30 seconds. The current-limited output stages of the RS-232 drivers are exempt from this limitation.

FRONT PANEL CONTROL

The ICFI system hardware front panel controls (see figure 3) allow limited manual control of the fault injection process. The manual control is restricted to the selection or enabling of:

1. any of the first 8 output probes (maximum of 2 simultaneously enabled probes)

2. SA1 or SA0 fault injection on the enabled probes or

3. external signal control (SA1, SA0, or square wave fault types) of the enabled probes.

The front panel switch settings determine the mode in which the ICFI system will work. These switches are (as labeled in the front panel) power, on/off line, relays, enables, complements, and pulses. The function of the power button is obvious, when "on" it lights up and so does a set of LED's which monitor the presence of all the voltage references and power supplies in the ICFI hardware. These LED's are labeled after the voltage they monitor and if any one fails to light up it signals a failure of the respective power supply.

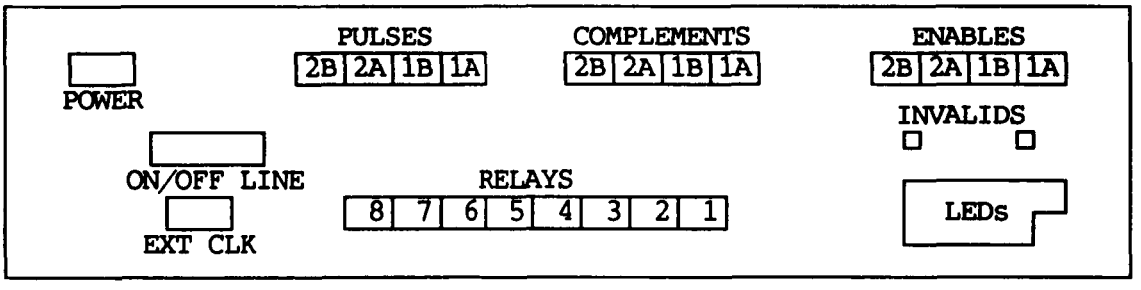

Figure 3. ICFI system front panel diagram.

The on/off line button setting determines the ICFI fault injection control mode, automatic (from the VAX) or manual (front panel). If set on-line the system can be controlled by software through the device driver interface (see section II) and the front panel's "Relays", "Enables", "Complements", and

"Pulses" (RECP) switches are disabled. If set off-line the front panel RECP switches are enabled and the software cannot be used to enable output probes or to inject a fault. The voltage levels V_+ and V_- have to be set through software in both on-line or off-line settings (see section II).

With the RECP switches enabled it is possible to manually control a fault injection. The "relays" switches labeled 1 through 8 enable the corresponding output probes. The "enables" switches in conjunction with the "complements" switches are used to inject SA1 or SA0 faults in the enabled output probes (see table 5). A pair of LED's below the RECP switches (labeled INVALIDS in fig. 3) signal when an illegal combination of "enables", "complements", and "pulses" switches has been set. There are four switches (labeled: 1A, 1B, 2A, and 2B) in each "enables", "complements", and "pulses" group, of these four, only the ones labeled "1A" and "1B" are used. The switches labeled "2A" and "2B" are not functional. The order in which the switches are set does not matter. The fault appears at the probes's output as soon as the last switch required is set "on".

As mentioned above an external input (TTL compatible) is available on the front panel (EXT CLK in fig. 3) to control fault injections on the enabled probes. There are three possible fault types available with the external input, each selectable by an appropriate combination of switches. The first two possible fault types are SA1 and SA0 (for the switch settings see table 5). In this mode a logic one in the external input causes a fault (SA1 or SA0) to be injected on the enabled probes. A logic zero clears the fault. For the third fault type available, square wave, a logic one (logic zero) on the external input injects a SA1 (SA0) on the enabled probes. Note that a SA1 (SA0) fault is at V_+ (V_-) volts.

TABLE 5. – FRONT PANEL SWITCHES SETTINGS FOR MANUAL FAULT INJECTIONS[a]

fault type	switches					
	enables		complements		pulses	
	1A	1B	1A	1B	1A	1B
SA0	off	on	off	on	off	off
SA1	on	off	on	off	off	off
external input(SA0)	off	on	off	off	off	on
external input(SA1)	on	off	off	off	on	off
external input square wave	on	on	off	on	on	on

[a]A switch is "on" in the "up" position.

Some applications of the manual fault injection capability are:

1) to test the probes (i.e., verify correct functionality)

2) to inject one or two faults at selected locations to verify some unusual behavior of the fault tolerant computer, e.g., the computer fails to detect a fault injection. By doing this it can be corroborated that the observed behavior is not caused by a malfunction of the ICFI system.

PROBE-CIRCUIT UNDER TEST (CUT) CONNECTION

The ICFI system output probes connect directly to the CUT through a dip clip. The output probes are grouped in modules of 4 probes each. Each module has 6 wires: V_{cc} (red), V_{ee} (black), and 4 probes (black, labeled with the probe number). V_{cc} has to be connected to the positive supply of the chip (e.g., +5 volts for TTL chips) and V_{ee} to the negative supply (or ground in TTL logic). When connecting the probes to the CUT with the ICFI system power on, the following procedure should be followed:

1. make sure the ICFI system is disabled, i.e., is not currently injecting a fault (if in manual control mode, make sure all switches are set off)

2. connect V_{ee} (black wire, unnumbered) first

3. connect V_{cc} (red wire) next

4. connect the probes to the desired pins.

REFERENCES

1. VAX/VMS MACRO User's Guide
   Digital Equipment Corporation, Maynard, MA

2. VAX/VMS System Services Reference Manual
   Digital Equipment Corporation, Maynard, MA

3. VAX/VMS Guide to Writing a Device Driver
   Digital Equipment Corporation, Maynard, MA

4. C.R. Elks, D.F. Green, D.L. Palumbo: User's Guide to Programming Fault Injection and Data Acquisition in the SIFT Environment. NASA TM-87638, 1987.

5. ECL Logic and Memory Data Book
   HITACHI, Ltd., Tokyo, Japan

6. The TTL Data Book, Vol. 2, 1985
   Texas Instruments, Inc., Dallas, Texas

7. CMOS Data Book, 1981
   National Semiconductor Corp., Santa Clara, CA

8. IC Master, Vol. II, 1986
   Hearts Business Communications, Inc., Garden City, NY

Standard Bibliographic Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
NASA TM-100478		

4. Title and Subtitle	5. Report Date
In-Circuit Fault Injector User's Guide	June 1987
	6. Performing Organization Code

7. Author(s)	8. Performing Organization Report No.
Peter A. Padilla	

9. Performing Organization Name and Address	10. Work Unit No.
NASA Langley Research Center Hampton, VA 23665-5225	505-66-21-07
	11. Contract or Grant No.

12. Sponsoring Agency Name and Address	13. Type of Report and Period Covered
National Aeronautics and Space Administration Washington, DC 20546-0001	Technical Memorandum
	14. Sponsoring Agency Code

15. Supplementary Notes

16. Abstract

A fault injector system, called in-circuit injector, was designed and developed to facilitate fault injection experiments performed at NASA-Langley's Avionics Integration Research Lab (AIRLAB). The In-Circuit Fault Injector (ICFI) allows fault injections to be performed on electronic systems without special test fixtures, e.g., sockets. The system supports stuck-at-zero, stuck-at-one, and transient fault models.

The ICFI system is interfaced to a VAX-11/750 minicomputer. An interface program has been developed in the VAX. The computer code required to access the interface program is presented. Also presented, is the connection procedure to be followed to connect the ICFI system to a circuit under test and the ICFI front panel controls which allow manual control of fault injections.

17. Key Words (Suggested by Authors(s))	18. Distribution Statement
In-Circuit Fault Injector	Unclassified-Unlimited Subject Category 33

19. Security Classif.(of this report)	20. Security Classif.(of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	22	A02

For sale by the National Technical Information Service, Springfield, Virginia 22161