DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF SCIENCES AND HEALTH PROFESSIONS
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA

MONITORING AND QUEUING FOR SIFT

By

Larry Wilson, Principal Investigator

Final Report
For the period January 1 to May 15, 1983

Prepared for the
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia

Under
Master Contract Agreement NAS1-17099
Task Authorization No. 11
Daniel Palumbo, Technical Monitor
Flight Control Systems Division

September 1983

DEPARTMENT OF COMPUTER SCIENCE
SCHOOL OF SCIENCES AND HEALTH PROFESSIONS
OLD DOMINION UNIVERSITY
NORFOLK, VIRGINIA

MONITORING AND QUEUING FOR SIFT

By

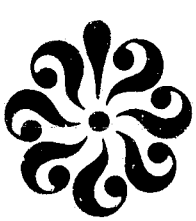
Larry Wilson, Principal Investigator

Final Report
For the period January 1 to May 15, 1983

Prepared for the
National Aeronautics and Space Administration
Langley Research Center
Hampton, Virginia

Under
Master Contract Agreement NAS1-17099
Task Authorization No. 11
Daniel Palumbo, Technical Monitor
Flight Control Systems Division

Submitted by the
Old Dominion University Research Foundation
P.O. Box 6369
Norfolk, Virginia 23508



September 1983

PART I

INSTRUMENTATION SOFTWARE FOR SIFT

INSTRUMENTATION FOR SIFT

CONTENTS

SUMMARY OF WINDOW COMPONENTS - SENDER AND RELAY

The SIFT instrumentation will be called the "Window." This window has been designed to collect internal data from SIFT while having minimal overhead.

Window consists of Sender and Relay components. Sender is to be run on processors 0..5 and Relay will run on processor 6. Sender will gather values (currently 12) during the subframe allocated to a task and broadcast these values at the start of the next subframe. This timing was selected to guarantee Relay 3.2ms to collect and transmit the data.

It is anticipated that Sender will require 50-80us during each subframe. Also, a task, which tries to use the broadcast bus early in its execution and which runs in a subframe with no vote, will have to delay until approximately 228us after Sender finishes.

The last value broadcast by Sender will always be taskid. After noting a change in taskid from two working processors, Relay will delay for synchronization and then gather the data from Sender into array WDATA. While gathering the data, Relay will add processor numbers and selected task output to it. Relay will then transmit the data from SIFT via the routine WDUMP. Relay will maintain a current value of the SIFT variable used for reconfiguration in order to distinguish working processors. Relay must recycle every 3.2ms and will produce data at a maximum rate of 84 words per 3.2ms.

IMPLEMENTATION OF SENDER

1. ADD GLOBAL

```
CONST WNOS=11;      (*WINDOW WILL COLLECT DATA 0..WNOS*)
      MAXPROCESSOR=5;  (*CHANGE FROM *)

VAR HOOK0,HOOK1,HOOK2,
    HOOK3,HOOK4        (*HOOKS*)
```

(* If we wish to monitor values which are not normally stored in SIFT, then we will use hooks. If we do not use all of the hooks to monitor local executive values, we may assign each of the remaining hooks to carry the value of a different variable from each task.. These assignments will be done in the PASCAL code of either the tasks or the local executive. We are not restricted to just 5 hooks since we can insert any number of hooks and change the addresses in pane (using AFTI) to dynamically select the ones to output. We may also choose to output the values of any of SIFT's global variables or constants by inserting their address into pane (without hooks).

The overhead of Window will increase by 3-8us for each hook used.*)

2. INSERT PROCEDURES SENDER AND INITSENDER. CALL SENDER AFTER A CLOCK INTERRUPT IN SCHEDULER AND CALL INITSENDER FROM INITIALIZE.

3. OVERHEAD TIME OF WINDOW PER SUBFRAME

   A. PROCEDURE SENDER- 30us
   B. POSSIBLE BROADCAST DELAY- 228us
      WILL ONLY OCCUR FOR A TASK WHICH BROADCASTS IMMEDIATELY AND FOLLOWS A TASK WHICH DOES NOT PRODUCE A VOTE.
   C. EXTRA TIME FOR HOOKS USED. 3-8us

4. FREE BUFFERS 0..11 FROM OTHER USES. IN PARTICULAR CLOCKTASK,IC1, WORK AND SYNCH WILL BE AFFECTED.

DYNAMIC MODIFICATION OF THE VARIABLES MONITORED

Sender is written to collect the values to be broadcast by using indirect addressing with the addresses stored in the location Pane (*which behaves as a Pascal array [0..11] of pointers*). By modifying the contents of pane via AFTI, the user can change the variables monitored.

RELAY COMMENTS

1. Relay will probably not be implemented as written. Current thinking is to use the AFTI read capability to remove the data from processor 6. Some of the ideas in Relay may prove valuable in writing code for this process. If the AFTI is not used, then Relay should be ready to go with minor changes to adapt to a 1553a or a parallel bus to the Vax.
2. The code used in Relay may take too long to execute (approximately 2ms). This time must combine with the transmission time to a total of less than 3.2ms. If necessary to shorten the time of Relay either cut back on the use of arrays or go into the assembler code to reduce the execution time.

OUTPUT DESCRIPTION

The output per subframe per processor will be an array 0..14 of integers containing the following data:

1. PROCESSOR ID
2. W0
3. W1
4. W2
5. W3
6. W4
7. ERRORS OF VP[0]
8. ERRORS OF VP[1]
9. ERRORS OF VP[2]
10. ERRORS OF VP[3]
11. ERRORS OF VP[4]
12. ERRORS OF VP[5]
13. TASKID
14. TASKOUTPUT

The contents of 2..12 may be changed dynamically by modifying the addresses in pane by using AFTI. The contents of 14 may be changed by interrupting processor 6 after the procedure INIT relay has been run and changing the addresses in the array TTOV by using the AFTI.

The output per subframe if all processors are working will be an array 0..5 of processor output. This will produce 84 words of output per subframe and this will take the following form:

```
DATA OF PROCESSOR  0
DATA OF PROCESSOR  1
DATA OF PROCESSOR  2  *
DATA OF PROCESSOR  3
DATA OF PROCESSOR  4
DATA OF PROCESSOR  5
```

*If processor 2 (for example) is not in the current configuration, then its data will not be present. Thus the data produced by processors 3..5 will all move up to fill the gap. Thus by looking at the data present we can identify which processors were working and we can calculate the virtual processor number of each real processor during this subframe.

The output per frame will be an array 0..26 (current schedule) of subframe output. This will total approximately 2K words of data for every 100ms. If storage becomes a problem, we could selectively block the output from some or most of the tasks from passing through Window or we could have the selectively ignore part of the data produced.

INITSENDER CODE

```
Procedure INITSENDER:
(*Set up the transfile to handle Sender broadcasts*)

Var B, Bend, TP:Integer:

BEGIN

     B:= 0;
     BEND: = B + WNOS;
     While  B < BEND DO
       BEGIN
         TP: = B + TPBASE;
         Transfile [2*TP - 1023]: = B*8;
         B: = B + 1
       END;
     TP: = B + TPBASE;
     Transfile [2*TP - 1023]: = Eofbit bor (Bend*8);
END;
```

SENDER CODE

```
      EXTRN  HOOK0
      EXTRN  HOOK1
      EXTRN  HOOK2
      EXTRN  HOOK3
      EXTRN  HOOK4

      EXTRN  ERROR
      EXTRN  TASKID
      EXTRN  TRANSP
      EXTRN  PIDEO

      ENTRY  SENDE


PANE  LINK   HOOK0
      LINK   HOOK1
      LINK   HOOK2
      LINK   HOOK3
      LINK   HOOK4

      LINK   ERROR       *Error is now indexed by VIRTNOS.  Be careful
      LINK   ERROR + 1   *Must coordinate with reconfig information to
      LINK   ERROR + 2    avoid errors in interpretation.
      LINK   ERROR + 3
      LINK   ERROR + 4
      LINK   ERROR + 5
      LINK   TASKID
```

*Comments - The addresses in PANE[I] 0<=I<11 may be modified dynamically. The current design of relay requires the address of taskid to remain in PANE[11]

```
ATRAN LINK   TRANSPTR
APIDE LINK   PIDEOF
WLOC  FIX    30592   ADDRESS OF DATAFILE[TPBASE+16#7400]
APANE LINK   PANE
```

```
SENDE PUSHM    0,3

      TRA    3,15
      LOAD   0,WLOC
      LOAD   1,APANE

      LOAD*  2,0,1          LOAD ONE
      STO    2,0,0     STORE ONE  REPEAT 12 TIMES
      LOAD*  2,1,1
      STO    2,1,0
      LOAD*  2,2,1
      STO    2,2,0
      LOAD*  2,3,1
      STO    2,3,0
      LOAD*  2,4,1
      STO    2,4,0
      LOAD*  2,5,1
      STO    2,5,0
      LOAD*  2,6,1
      STO    2,6,0
      LOAD*  2,7,1
      STO    2,7,0
      LOAD*  2,8,1
      STO    2,8,0
      LOAD*  2,9,1
      STO    2,9,0
      LOAD*  2,10,1
      STO    2,10,0
      LOAD*  2,11,1
      STO    2,11,0

WAIT  LOAD*  1,APIDE          WAITBROADCAST
      SKEQ   1,WAIT
      LOAD   1,WLOC
      STO*   1,ATRAN        BEGIN THE BROADCAST
      TRA    15,3
      POPM   0,3

      RPS    0
```

RELAY DESCRIPTION

After initialization, the Program Relay repeats the following loop:

Reconfigure if necessary

Collect the data

Output the data (or be read)

Calculate configuration if necessary

RELAY - APPROX. TIMING

Wreconf - 16 + (24*6)	160us
Wcollect - 8 + (2*Search) + delay	1,596us
+ move where	
search - 4 + (26*6) = 160us	
delay - 100us	
move - 4 + 6*(13 + 912*13) + 15 + 10)	
= 1,168	
Wdump	?
gexec	62
Relay	20
Total	1,838 + ?

RELAY CODE

```
Program Relay;                          (*Processor 6 only*)

Const   MAXPROCESSOR = 5;
        GET          = 3;               (*Global executive task is #3*)
        WNOS         = 11;
        datnum       = 84;              (*Max data output per subframe*)
        dloc         = 16#7400;
        clkloc       = 16#77FD;
        TTOVLOC      = 16#4000;         (*Taskid to variable location*)
        ERRER        = 33;              (*Mandatory SIFT buffers*)
        gexecreconf  = 34;
        gexecmemory  = 35;
        expected     = 36;
        lock         = 37;
        ndr          = 38;
        XRESET       = 39;              (*Probably should include all Buffers*)
        qcmdail      = 103;
        qcmdele      = 104;
        qcmdrud      = 105;
        qcmdthr      = 106;
        qdelv        = 107;
        qfwda        = 108;
        qpitmo       = 109;
        qiatmo       = 110;
        qreconf      = 111;
        olast        = 111;             (*must correspond to last of O series*)
        osynch       = 112;
        (*internal values*)
        phin         = 113;
        psin         = 114;
        rn           = 115;
        qx           = 116;
        qy           = 117;
        qz           = 118;
        timer        = 119;
        Maxdata      = 1015;
        dbsize       = 128;
        Maxtime      = 16 # 47;

Type    Procint : array [0..Maxprocessor] of integer;
        Procbool: array [0..Maxprocessor] of boolean;
        dfindex : 0.. Maxdata;

Var     clock at clkloc: integer;
        first,                          (*alphabetic I hope*)
        ibase,
        lastconfig,
```

```
        next,
        NW,
        Reconfig,
        Taskid: integer
        datafile at dfloc: array [dfindex] of integer;
        dbad: Procint;
        oldtask: Procint;
        TTOV at TTOVLOC: array [0..11] of integer; (*Taskid to variable index*)
        Wdata: arry [0..datnum] of integer;
        working: Procbool;
        Vtor: Procint;
        Vtodf: Procint;

Procedure Initializerelay;
  (* In Initialize
  1.  We assume all processors are working.  If this is not true at the start,

      we will collect some garbage output until a reconfiguration occurs in

      SIFT.

  2.  Unless Relay is initialized and waiting before SIFT finishes

      initialization, the early data may be garbled because of a race

      condition.  This will clear up after IC3 first runs in SIFT, since it

      will allow Relay time to catch up. *)

VAR i, ad: Integer;

BEGIN   ad:= 0;
        For i:= 0 to maxprocessor do
          BEGIN
            oldtask [i]:= datafile [ad + WNOS];
            dbad [i]:= ad;
            ad:= ad + dbsize
          end;
        Reconfig:= 0; (*all working*)
        Lastconfig:= 1; (*trigger a reconfiguration*)
        (*Initialize the index to select an output from each task*)

              (*Taskid to variable location*)
        TTOV[0]:= 11; (*repeat Taskid why not?*)
        TTOV[1]:= 11;
        TTOV[2]:= ERRER;
        TTOV[3]:= Gexec;
        TTOV[4]:= 11;
        TTOV[5]:= 11;
```

```
        TTOV[6]:= Expected;
        TTOV[7]:= 11;
        TTOV[8]:= 11;
        TTOV[9]:= QX;
        TTOV[10]:= QLATMO;
        TTOV[11]:= QPITM;
        TTOV[12]:= CMDRU
END;

Procedure WReconfigure;
VAR s,i: Integer;
Begin   s:= Reconfig;
        Lastconfig:= s;          (*lets not do this again soon*)
        NW:= -1;
        i:= 0;
        Repeat
          If odd(s)
            then working [i]:= false;
            ELSE
              Begin
                Working [i]:= true;
                NW:= NW + 1;
                (*Vtor [NW]:= i *)
                (*Vtodf [NW]:= dbad[i]*)
              END;
          s:= s div 2;
          i:= i + 1
        Until i > Maxprocessor
END;

Procedure WCollect;              (*wcollect calls Search twice and moveit
                                  once*)
  VAR wclock: Integer;

Procedure Search
  VAR i: Integer;
  (*find a processor not equal to first who has changed his taskid.  Search is
called twice by Wcollect*)
  Begin  (*could be coded with Vtor and Vtodf*)
        i:= -1
        Next:= -1;
        Repeat
          i:= i + 1 MOD 6;
          if working [i] then
            if i <> first then
              if oldtask [i] <> datafile [dbad[i] + WNOS]
                then Next:= i
        Until Next:= i
  END;  (*Search*)
```

```
Procedure Moveit;
  VAR i,j,offset,newoff: Integer;
  Begin (*Moveit*)
    offset:= -1;
    For i:= 0 to Maxsender DO
      if working [i] then
        Begin
          offset:= offset + 1;
          Wdata [offset]:= i;  (*Store processor number*)
          i:= 0
          ibase:= dbad [i];
          Repeat                (*fetch and store broadcast values from window*)
            offset:= offset + 1;
            Wdata [offset]:= datafile [ibase + j]
            j:= j + 1
          Until j = WNOS + 1;
          taskid:= Wdata [offset];
          oldtask [i]:= Taskid;
          offset:= offset + 1;
          Wdata [offset]:= datafile [TTOV[taskid] + ibase]
        end (*if working*)
  End; (*Move*)
  Begin (*WCollect*)
    first:= -1;
    Search;                        (*find first change in taskid*)
    first:= Next;
    Search;                        (*find second change in taskid*)
    wclock:= clock;                (*delay for clockskew*)
    while clock - wclock < Maxtime do;
    Moveit
  END;

Procedure WDump;
  Make connection to send
  [14*(NW + 1) words] from the start of wdata
  to Vax Via:
            1)Sox box - 2.8ms - can be reduced
            2)1553a bus - 1.6ms
            3)parallel bus  a. .84ms no handshake
                            b. .28ms with handshake

  * all times are approximations
```

```
Procedure Wgexec;
VAR i,j: Integer;
      r: array [0..2] of integer;
Begin
  i:= 0;
  j:= 0;
  Repeat     (*See if three or more processors ran Rect in SIFT*)
             (*we choose to not reconfigure for two processors*)
    If old task [i] = get then
    If working [i] then
      Begin
        r[j]:= datafile [dbadd[i] + Gexec];
           j:= j + 1
      END;
    i:= i + 1
  Until ( i > Maxprocessor) or ( i > 2);
  If j ≥ 2 then
    if r[1] = r[2] then reconfig:= r[1]
      ELSE reconfig:= r[0]
END;     (*default of vote, we do not have to be fault tolerant*)

Begin (*Relay*)
  Initializerelay;
  While true do Begin
    if Lastconfig <> Reconfig then WReconfig;
    Wcollect;
    Wdump;
    Wgexec;
  END (*while true*)
END (*Relay*)
```

PART II

INTERNAL COMMUNICATION IN SIFT

INTERNAL COMMUNICATION IN SIFT

The interprocessor communications are constrained by time and by the size of memory available for broadcasting. A worst case broadcast time requires approximately 19us per word. When the data collection component of the instrumentation package is installed, it will consume a portion of the broadcast area as well as increasing the probability of a delay due to the serial nature in which broadcast requests are honored. This delay will always be less than 228 us (19 x 12 = 228) and it is anticipated that any critical delays can be partially remedied by delaying the requests for broadcasts in the troubled tasks.

The current broadcast protocol restricts the variables to 120 datafile locations (*1024 : 8 minus some dedicated locations*). This could be increased to about 170 by gearing the system to 6 processors rather than 8. This is not a permanent solution unless 6 is determined to be an optimal number of processors in the system. We could also increase the number of variables transmitted by the system by multiplexing the datafile area. This will slow the system down and/or have the negative effect of making the name of a memory location time dependent. We prefer to speed up SIFT and to keep the design as simple as possible. The descendents of SIFT will be equipped with much larger datafiles, thus multiplexing is not likely to be necessary.

The other problem associated with internal communications is the delay called waitbroadcast. If the bus is busy when requested, a processor may have a busy wait (wasted time) until the bus becomes available. The implementation of a broadcast queue could alleviate these time wastes. This could either be done using the datafile receive area for the eighth processor or in the current broadcast area. This feature could have some value to future SIFT-like machines and could be implemented and tested against baseline SIFT at this time. This would require 4-6 manweeks.

The future uses of SIFT will probably include nonpriority tasks with replicates running in scattered subframes. This will create the problem of consistency of input for nonpriority tasks. Each such task will require multiple buffering of certain postvote buffer values used for input. These input values can be updated only between the completion of the task's last replicate and the start of the first replicate on the next iteration.

I recommend the broadcast queue as a potential improvement to baseline SIFT which can be implemented and tested at this time. The multiple buffering should wait for new developments in scheduling. Multiplexing is not as desirable at this time.