NOTICE

THIS DOCUMENT HAS BEEN REPRODUCED FROM MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED IN THE INTEREST OF MAKING AVAILABLE AS MUCH INFORMATION AS POSSIBLE.
FUNCTIONAL SPECIFICATIONS OF THE ANNULAR SUSPENSION POINTING SYSTEM

by

Bryan Edwards
Department of Computer Science
University of Colorado at Boulder
Boulder, Colorado

CU-CS-171-80
January, 1980

This work was supported by grant NSG 1638 from NASA Langley Research Center.
Abstract

The following is a description of the Annular Suspension Pointing System. This description is written using the Design Realization, Evaluation and Modelling (DREAM) system, and its design description technique, the DREAM Design Notation (DDN).
Appendix A contains a DON description of the Annular Suspension Pointing System. The information contained in this description was derived from the NASA-produced report, "The Executive Software For the Annular Suspension Pointing System," which appears as Appendix B. The description is divided into four major sections.

The first section of Appendix A (System Overview) contains the major units of the system, their interconnections, and the event flow between these units. Figure 1 corresponds to Figure 1 in the original report, with the addition of three major units: analog sources, experiment computer, and the system operator. Additional communication paths are also shown. Each communications path is labeled with a number. These numbers correspond to the CONNECTIONS given in the DREAM description. In addition, the EVENT DEFINITIONS reference the communications paths which the events use, by appending the path number to the event name. Finally, the legal event sequences are given in the DESIRED BEHAVIOR section of the description, using a regular expression type notation. In this section, a shorthand, non-standard, notation is used to indicate the repetition of a sequence of events a specific number of times.

The second section (LEVEL II) describes the basic operations of each of the major units of the system. The input and output ports are identified, and an abstract model of the operation is given in terms of the input and output.

In the third section (LEVEL III), the notion of the internal servicers P(1), P(2) and P(3) is introduced. The internal operation of these servicers is not detailed. The logical interaction between the servicers and the input and output ports of the NASA standard space computer is given.

The notion of the time intervals T(1), T(2) and T(3) is introduced in the fourth section (LEVEL IV). Here we see the interaction between the master timing pulse and the signals to the three processes P(1), P(2) and P(3).

In Appendix B, we have included a copy of the NASA-produced functional specification of the Annular Suspension Pointing System.

The portions of the report which were captured by the DON description of the system are underlined.

Some portions of the NASA report contain very detailed descriptions of sections of the system. This detail is not reflected in the DON description. Further elaborations of the DON description would be required to capture this detail.

The DON description does not capture the notion of the mode (idle, coarse, fine, slew) of the system, and the details of the data communicated between the system units is not given. The NASA report does not contain enough information in these areas to allow further elaboration.

In preparing the DON description, the concepts available in DON were adequate to describe most of this embedded computer system. The only area that DON does not adequately describe is the notion of a specific interval of time.
FIGURE 1
Communications paths between system components

Appendix A:
DREAM Design Notation Description
of Annular Suspension Pointing System
The purpose of the APSS is to control a platform which will be flown on the space shuttle. Equipment (e.g., a telescope) will be mounted on the platform and the APSS will allow this equipment to be pointed in a given direction with extreme accuracy ($4.84 \times 10^{-7}$ radians) and this position maintained for extended periods (stability $4.84 \times 10^{-7}$ radians/sec) in the presence of shuttle disturbances.
EVENT DEFINITION:

- system_operator.request_experiment_19: DESCRIPTION;
  This event corresponds to the system operator entering a request at the operator console to the experiment computer.
  END DESCRIPTION;

- experiment_computer.request_platform_action_17,13: DESCRIPTION;
  In order to perform a given experiment, the experiment computer must manipulate the platform in some predefined manner.
  END DESCRIPTION;

- dea.request_computation_3: DESCRIPTION;
  Many times, computations must be performed before a requested platform action can occur.
  END DESCRIPTION;

- nssci.computation_result.returned_1: DESCRIPTION;
  Computations supporting the platform are performed in the nssci. Results are returned to the dea.
  END DESCRIPTION;

- dea.request_platform_action_7,11: DESCRIPTION;
  The platform is actually controlled by the pea/cea.
  END DESCRIPTION;

- platform.responds_21,10: DESCRIPTION;
  The platform responds to request from the pea/cea.
  END DESCRIPTION;

- platform.result.returned_to_experiment_computer_8,14: DESCRIPTION;
  The results of a high-level platform operation are returned to the experiment computer.
  END DESCRIPTION;

- experiment_result.returned_to_system_operator_18: DESCRIPTION;
  The result of our experiment is returned to the system operator at the operator console.
  END DESCRIPTION;

- system_operator.request_test_20: DESCRIPTION;
  This event corresponds to the system operator entering a request at the test console to the test support equipment.
  END DESCRIPTION;

- tse.request_action_15: DESCRIPTION;
  In order to perform a given test, the test support equipment must get certain data from the nssci.
  END DESCRIPTION;

- nssci.result.returned_2: DESCRIPTION;
  Test data from the nssci is returned to the test support equipment.
  END DESCRIPTION;

- test.result.returned_to_system_operator_16: DESCRIPTION;
  The result of a test is returned to the system operator at the test console.
  END DESCRIPTION;

- master_timing_pulse_9: DESCRIPTION;
  This pulse is generated every T(1) milliseconds.
  END DESCRIPTION;

- tl_timing_pulse_4: DESCRIPTION;
  This pulse is generated every T(1) milliseconds as a result of the master timing pulse.
  END DESCRIPTION;

- t2_timing_pulse_5: DESCRIPTION;
  This pulse is generated every T(2) milliseconds as a result of the master timing pulse. Note that T(2) milliseconds is an integral multiple of T(1).
  END DESCRIPTION;
SYSTEM OVERVIEW

t3_timing_pulse_6: DESCRIPTION;
This pulse is generated every T(3) milliseconds as a result of the master timing pulse. Note that T(3) milliseconds is an integral multiple of T(2).
END DESCRIPTION;

DESIRED BEHAVIOR;

SHUFFLE(
  REPEAT(
    SEQUENCE(
      SEQUENCE(
        SEQUENCE(master_timing_pulse_9, tl_timing_pulse_4), t1 per t2
        t2_timing_pulse_5), t2 per t3
        t3_timing_pulse_6)),
    REPEAT(
      SEQUENCE(
        system_operator_request_experiment_19,
        REPEAT(
          SEQUENCE(
            experiment_computer_requests_platform_action_17,13,
            REPEAT(
              SEQUENCE(
                (dea_request_computation_3,
                  nssci_computation_result_returned_1)),
              REPEAT(
                (dea_request_platform_action_7,11,
                  platform_responds_21,10)),
                platform_result_returned_to_experiment_computer_8,14)),
              experiment_result_returned_to_system_operator_18)),
            REPEAT(
              sequence(
                system_operator_request_test_20,
                REPEAT(
                  sequence(
                    (tse_request_action_15,
                      nssci_results_returned_2)),
                  test_result_returned_to_system_operator_16))
          ))
      ))
  ))
)

END DESIRED BEHAVIOR;

END EVENT DEFINITION;

END EVENT CLASS;

END SUBSYSTEM CLASS;
LEVEL II

system operator: SUBSYSTEM CLASS;

ec outputs: OUT PORT;
END PORT;

tse outputs: OUT PORT;
END PORT;

et inputs: IN PORT;
END PORT;

tse inputs: IN PORT;
END PORT;

operator: CONTROL PROCESS;

MODEL;

ITERATE

SELECT
(Perhaps): SEND ec outputs;
(Perhaps): SEND tse outputs;
END SELECT;

SELECT
(Perhaps): RECEIVE ec inputs;
(Perhaps): RECEIVE tse inputs;
END SELECT;

ITERATE;

END MODEL;

END CONTROL PROCESS;

END SUBSYSTEM CLASS;

LEVEL II

[experiment_computer]: SUBSYSTEM CLASS;

rau outputs: OUT PORT;
END PORT;

so outputs: OUT PORT;
END PORT;

rau inputs: IN PORT;
END PORT;

so inputs: IN PORT;
END PORT;

experiment: CONTROL PROCESS;

MODEL;

ITERATE

RECEIVE so_inputs;
SEND rau_outputs;
SEND rau_inputs;
SEND so_outputs;

END ITERATE;

END MODEL;

END CONTROL PROCESS;

END SUBSYSTEM CLASS;
LEVEL II
[remote acquisition unit]: SUBSYSTEM CLASS:
  dea outputs: OUT PORT;
  END PORT;
  ec outputs: OUT PORT;
  END PORT;
  dea inputs: IN PORT;
  END PORT;
  ec inputs: IN PORT;
  END PORT;
  acquisition: CONTROL PROCESS;
  MODEL;
  ITERATE
    RECEIVE ec inputs;
    SEND dea outputs;
    RECEIVE dea inputs;
    SEND ec outputs;
  END ITERATE;
  END MODEL;
  END CONTROL PROCESS;
END SUBSYSTEM CLASS;

LEVEL II
[platform]: SUBSYSTEM CLASS:
  pea_cea outputs: OUT PORT;
  END PORT;
  pea_cea inputs: IN PORT;
  END PORT;
  platform: CONTROL PROCESS;
  MODEL;
  ITERATE
    RECEIVE pea_cea inputs;
    SEND pea_cea outputs;
  END ITERATE;
  END MODEL;
  END CONTROL PROCESS;
END SUBSYSTEM CLASS;
[analog_sources]: SUBSYSTEM CLASS;
  dea outputs: OUT PORT;
  END PORT;
  source: CONTROL PROCESS;
  MODEL:
    ITERATE
      SEND dea outputs;
    END ITERATE;
  END MODEL;
END CONTROL PROCESS;
END SUBSYSTEM CLASS;

[test_support_equipment]: SUBSYSTEM CLASS;
  nsscl outputs: OUT PORT;
  END PORT;
  so outputs: OUT PORT;
  END PORT;
  nsscl inputs: IN PORT;
  END PORT;
  so inputs: IN PORT;
  END PORT;
  support: CONTROL PROCESS;
  MODEL:
    ITERATE
      RECEIVE so inputs;
      SEND nsscl outputs;
      RECEIVE nsscl inputs;
      SEND so outputs;
    END ITERATE;
  END MODEL;
END CONTROL PROCESS;
END SUBSYSTEM CLASS;
LEVEL II
[platform electronics assembly control electronics assembly]: SUBSYSTEM CLASS:

master timing pulse: OUT PORT;
END PORT;
dea outputs: OUT PORT;
END PORT;
p outputs: OUT PORT;
END PORT;
dea inputs: IN PORT;
END PORT;
p inputs: IN PORT;
END PORT;
control: CONTROL PROCESS;
MODEL;
ITERATE
SELECT
(P: PAPS): SEND master timing pulse;
(PEAPS): RECEIVE dea inputs;
SEND  p outputs;
RECEIVE  p inputs;
SEND  dea outputs;
END SELECT;
END ITERATE;
END MODEL;
END CONTROL PROCESS;
END SUBSYSTEM CLASS:

LEVEL II
[nasa standard space computer ii]: SUBSYSTEM CLASS:
deaf outputs: OUT PORT;
END PORT;
tse outputs: OUT PORT;
END PORT;
deaf inputs: IN PORT;
END PORT;
t1 tick: IN PORT;
END PORT;
t2 tick: IN PORT;
END PORT;
t3 tick: IN PORT;
END PORT;
tse inputs: IN PORT;
END PORT;
mscii executive: CONTROL PROCESS;
MODEL;
perform initialization;
SEND dea outputs;
RECEIVE dea inputs;
ITERATE
SELECT
(PEAPS): RECEIVE t1 tick;
SEND  dea outputs;
(PEAPS): RECEIVE t2 tick;
(PEAPS): RECEIVE t3 ticks;
(PEAPS): RECEIVE tse inputs;
SEND  tse outputs;
END SELECT;
END ITERATE;
END MODEL;
END CONTROL PROCESS;
END SUBSYSTEM CLASS;
LEVEL II

[digital electronics assembly]: SUBSYSTEM CLASS:

nsccii outputs: OUT PORT;
   END PORT;

t1 tick: OUT PORT;
   END PORT;

t2 tick: OUT PORT;
   END PORT;

t3 tick: OUT PORT;
   END PORT;

nsccii outputs: OUT PORT;
   END PORT;

rau outputs: OUT PORT;
   END PORT;

nsccii inputs: IN PORT;
   END PORT;

master timing pulse: IN PORT;
   END PORT;

nsccii inputs: IN PORT;
   END PORT;

as inputs: IN PORT;
   END PORT;

rau inputs: IN PORT;
   END PORT;

---

LEVEL II

dea executive: CONTROL PROCESS:

MODEL:

ITERATE

SELECT

(Perhaps): RECEIVE master timing pulse;
   SEND t1 tick;
   SELECT
      (Perhaps) SEND t2 tick;
      (Perhaps) SEND t2 tick;
      SEND t3 tick;
   END SELECT;

(Perhaps): RECEIVE rau inputs;
   MAYBE
   SEND nsccii outputs;
   RECEIVE nsccii inputs;
   END MAYBE;
   MAYBE
   SEND rau inputs;
   RECEIVE rau inputs;
   END MAYBE;

(Perhaps): RECEIVE as inputs;
   END SELECT;

END ITERATE;

END MODEL;

END CONTROL PROCESS;

END SUBSYSTEM CLASS;
LEVEL III

'[nasa_standard_space_computer_ii]': SUBSYSTEM CLASS

SUBCOMPONENTS:
pl, p2, p3 OF [process]
END SUBCOMPONENTS;

CONNECTIONS;
PLUG (nsscii_executive|pl_initiate, pl|initiate),
PLUG (nsscii_executive|p2_initiate, p2|initiate),
PLUG (nsscii_executive|p3_initiate, p3|initiate),
PLUG (pl|complete, nsscii_executive|pl_complete),
PLUG (p2|complete, nsscii_executive|p2_complete),
PLUG (p3|complete, nsscii_executive|p3_complete)
END CONNECTIONS;

nsscii_executive: CONTROL PROCESS;
pl_initiate: LOCAL OUT PORT;
END PORT;
p2_initiate: LOCAL OUT PORT;
END PORT;
p3_initiate: LOCAL OUT PORT;
END PORT;
pl_complete: LOCAL IN PORT;
END PORT;
p2_complete: LOCAL IN PORT;
END PORT;
p3_complete LOCAL IN PORT;
END PORT;

MODEL:
perform initialization;
SEND dea outputs;
ITERATE
SELECT
(Perhaps): RECEIVE t1_tick;
SEND pl_initiate;
(Perhaps): RECEIVE t2_tick;
SEND p2_initiate;
(Perhaps): RECEIVE t3_tick;
SEND p3_initiate;
(Perhaps): RECEIVE pl_complete;
SEND dea_outputs;
(Perhaps): RECEIVE p2_complete;
(Perhaps): RECEIVE p3_complete;
(Perhaps): RECEIVE tse_inputs;
SEND tse_outputs;
END SELECT;
END ITERATE;
END MODEL;
END CONTROL PROCESS;
[process]: SUBSYSTEM CLASS;
  complete: OUT PORT;
  END PORT;
initiate: IN PORT;
  END PORT;
process: CONTROL PROCESS;
MODEL;
ITERATE
  RECEIVE initiate;
  perform_operations;
  SEND complete;
END ITERATE;
END MODEL;
END CONTROL PROCESS;
END SUBSYSTEM CLASS;

[digital_electronics_assembly]: SUBSYSTEM CLASS
QUALIFIERS:
  t1_per_t2, t2_per_t3
END QUALIFIERS;
LOCAL SUBCOMPONENT;
  t1_pulses OF [0:: t1_per_t2],
  t2_pulses OF [0:: t2_per_t3]
END LOCAL SUBCOMPONENT;
de_a_executive: CONTROL PROCESS;
MODEL;
  SET t1_pulses TO 0;
  SET t2_pulses TO 0;
ITERATE
  SELECT
    (PERHAPS): RECEIVE master_timing_pulse;
    SEND t1_tick;
    SET t1_pulses TO t1_pulses + 1;
    IF t1_pulses = t1_per_t2 THEN
      SET t1_pulses TO 0;
      SEND t2_tick;
      SET t2_pulses TO t2_pulses + 1;
      IF t2_pulses = t2_per_t3 THEN
        SET t2_pulses TO 0;
        SEND t3_tick;
      END IF;
    END IF;
    (PERHAPS): RECEIVE raw_inputs;
    MAYBE
      SEND nscii_outputs;
      RECEIVE nscii_inputs;
    END MAYBE
    MAYBE
      SEND pea_cea_outputs;
      RECEIVE pea_cea_inputs;
    END MAYBE;
    (PERHAPS): RECEIVE as_inputs;
  END SELECT;
END ITERATE;
END MODEL;
END CONTROL PROCESS;
This document attempts to describe the Annular Suspension Pointing System (ASPS) hardware facilities and the structure of the software executive in sufficient detail that it can be used as an example of the requirements for concurrent programming in NASA embedded computer systems. The hardware details are provided for those who are unfamiliar with the general layout of the ASPS. This description is intended to be accurate and every effort will be made to ensure that it correctly reflects the software currently being written for the ASPS engineering model. The engineering model is a ground-based system used for testing. This is the first version of the software which will be used. This description is also intended to be complete in the sense that the functions of the software is defined in sufficient detail (albeit informally) that only minor parametric details are needed before the software can be constructed.

The purpose of the ASPS is to control a platform which will be flown on the Space Shuttle. Equipment (e.g., a telescope) will be mounted on the platform and the ASPS will allow this equipment to be pointed in a given direction with extreme accuracy (+ or - 0.016 radians) and this position maintained for extended periods (stability + or - 0.049 radians per sec.) in the presence of Shuttle disturbances.

Appendix B: Functional Requirements of Annular Suspension Pointing System

(Produced by NASA Langley Research Center Personnel)
II. HARDWARE CONFIGURATION OF THE SYSTEM

Figure 1 shows the organization of the major hardware units comprising the ASPS.

Definitions:

a) TSE: Test Support Equipment. The TSE consists of a terminal and a computer. It will be used to generate various inputs from the operator and display messages to the operator for ground testing of the ASPS.

b) SSL II: NASA Standards Space Computer II. The SSL II is basically an IBM System 360 computer. It will be used to carry out the computations which implement the control laws for the platform.

c) PLA: Digital Electronics Assembly. The PLA is an electronics assembly based on a 2-KB microprocessor which is used as an I/O controller for the SSL II. Figure 2 shows the major hardware units of the PLA.

d) PLA/CLA: Platform Electronics Assembly/Control Electronics Assembly. The PLA is the electronic assembly on the platform which, together with the CLA, is responsible for moving the platform and sending position information to the SSL II.

e) RAU: Remote Acquisition Unit. The RAU provides 32 digital input and 32 digital output lines. It is connected to the experiment computer i.e., the experiment which is using the ASPS, and all I/O between the PLA and the experiment computer is through the RAU.

The system of interest for the executive consists of an SSL II computer connected to the PLA. The PLA is connected to the SSL II by 16 inputs, 16 output and several control lines. During flight all I/O to the SSL II is through the PLA. In the engineering model, I/O to

the SSL II can also be from the TSE. Analog data to or from the platform is converted (A/D, D/A) in the PLA and preprocessed in the PLA before being sent to the SSL II.

The experiment computer sends platform control commands (e.g., point telescope in a particular direction) to the PLA, which in turn requests the SSL II to compute the control laws. This output is sent through the PLA to the PLA/CLA which moves the platform.

The system master timing pulse is generated by the PLA and sent to the PLA. This is a pulse every T1 milliseconds which is used for real-time timing. T1 is a fixed integer whose value has not been finalized.

NOTE: This pulse is not sent directly to the SSL II.
III. ESSC II CHARACTERISTICS

This section provides a summary only and is not intended to be complete. Full details of the machine can be obtained from the hardware reference manual.

The IBM System/360 (ESSC II) is very much like an IBM System 360. A thorough knowledge of S/360 is assumed in this summary. For example, all instructions from the S/360 are included.

The ESSC II instruction set contains 83 of the 67 instructions from the S/360 Standard Set. (Recall that the Standard Set does not include decimal, direct control, protection or floating point instructions.) The exceptions are 110(9L), S10(9C), TCH(9F), and T10(9D). The semantics of these 83 instructions are identical to S/360 except for the following areas:

(1) The S/360 interval timer at location 80(decimal) is not implemented.

(2) Effective addresses are limited to 15 bits except for the LA(4I) instruction which generates a 24-bit result.

Added to these 83 instructions are three new instructions:

<table>
<thead>
<tr>
<th>Mnemonic</th>
<th>Opcode</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timer Read and Set</td>
<td>TURS</td>
<td>A4 RS</td>
</tr>
<tr>
<td>Start I/O</td>
<td>S10</td>
<td>A5 RS</td>
</tr>
<tr>
<td>Set Storage Key</td>
<td>SSE</td>
<td>08 RR</td>
</tr>
</tbody>
</table>

The ESSC II supports a real-time clock and an interval timer. The real-time clock is 32 bits long, is incremented by each 112.64 microseconds and causes no interrupt on overflow. The interval timer is 16 bits long, is decremented by each 112.64 microseconds and generates an external interrupt on change of sign from positive to negative.

The clocks are read, or read and set, individually by the U1PS instruction. Reading yields a timer value in a register. Setting involves a value from storage being placed in the timer.

There are four kinds of I/O. They are:

(a) Direct.
(b) Buffered.
(c) Direct Memory Access (DMA).
(d) External Interrupt.

Direct I/O constitutes transferring 16 bits of data to/from the ESSC II from/to the DLA via a 16-bit bus. Transfers of this type are the result of the ESSC II executing a S10 instruction. Note that this is totally different from the S10 instruction on S/360.

Buffered I/O is a means of performing block transfers of data to/from the ESSC II memory in parallel with normal execution of the ESSC II. When buffered I/O takes place, memory references and sequencing are controlled by hardware within the CPU but this does not interfere with instruction execution. The ESSC II has provision for up to 16 devices to perform buffered I/O. A fixed storage location is used on the ESSC II to point to a buffered I/O device table with 16 entries. An entry contains two words each of which is a word count and address pair. One word is for input and the other is for output. The word count is the number of words to be transferred and the address is the main memory buffer location. If the relevant word count is positive when a buffered I/O operation begins, then the count is decremented and the address is incremented in the table entry as each word is transferred. When the transfer is complete the word count will be zero (assuming no error). If the word count is initially negative, the word count is modified during the buffered I/O operation but is reset to its original value when the operation completes. The number of words transferred in this case is the absolute value of the word count. A buffered I/O operation is initiated using direct I/O (S10) instructions to send I/O
The ASPS executive has the primary goal of providing scheduling, in real time, of certain processes. There are three time periods of interest. At present, they are 10 milliseconds, 100 milliseconds and 1 second but these may be adjusted. These time periods will be referred to here by the symbols $T(1)$, $T(2)$, and $T(3)$, in millisecond time units. Associated with these time periods are three sets of processes. The set $(P(1,j))$ is associated with $T(1)$, the set $(P(2,j))$ with $T(2)$ and the set $(P(3,j))$ with $T(3)$. Certain computations must be completed every $T(1)$ milliseconds, others every $T(2)$ milliseconds and still others every $T(3)$ milliseconds. $T(2)$ and $T(3)$ are integer multiples of $T(1)$, and $T(3)$ is an integer multiple of $T(2)$.

Timing is centered around a 1/0 interrupt from the DLA which is derived from, but not coincident with, the system master timing pulse. This interrupt will be generated by the DLA every $T(1)$ milliseconds regardless of what the NSSC 11 does, although the NSSC 11 may mask it. At system start-up, the ASPS executive performs any data initializations which are necessary. Signals the DLA that initialization is complete using direct 1/0, and then places the NSSC 11 in the wait state with no processes active.

Processing begins when the first 1/0 interrupt arrives from the DLA. From then on, $P(1,j)$ must be completed every $T(1)$, $P(2,j)$ every $T(2)$ and $P(3,j)$ every $T(3)$ milliseconds of real time for some $j$. Real time can be thought of as a sequence of $T(3)$ time periods. Each $T(3)$ time period is broken into an integral number of $T(2)$ time periods, each $T(2)$ time periods is broken into an integral number of $T(1)$ time periods. The quantity $j$ is called the mode and a different process is used for each mode. The mode is the operating state of the platform and currently $k$ (i.e., the number of modes) is 4. They are called DLA, COURSE, FDL and SL. The system changes mode based on certain inputs (see below) and only certain transitions are valid. Mode changes can only occur at the beginning of a $T(3)$ millisecond time period.

The NSSC 11 interval timer is not used for any determination of real time. It is used solely as a check on the system master timing pulse. At the beginning of each $T(1)$ time period (i.e., following the interrupt from the DLA) the interval timer is loaded with a value slightly larger than $T(1)$. If the timer ever expires then clearly an error has occurred. For the initial version of the executive, if the timer interrupt ever occurs, the system will not attempt to recover but merely inform the operator and enter the wait state.

When the 1/0 interrupt at the beginning of $T(1)$ occurs the executive is entered. The DLA will already have completed a buffered input operation and placed a total of $L(10)$ words into the NSSC 11 memory. $L(10)$ is currently 38. This block of data is in two parts. The first

bits of data to the device which will perform the operation.

Direct memory access is literally direct access of the NSSC 11 memory. The ASPS system does not use DMA.

An external interrupt changes the state of the NSSC 11 as a result of an external stimulus and as such can be regarded as an input mechanism. The ASPS system does not use external interrupts for data input.

Memory on the NSSC 11 is protected in blocks of 1024 bytes. Storage keys are two bits long, and they are used for write protection only. One bit is used to inhibit CPU and buffered I/O storing and the other is used to inhibit DMA storing. Storage keys are set by the Set Storage Key (SSK) instruction, and also, following an interrupt, the storage key of the first block of memory is set to allow CPU and buffered I/O storing but inhibit DMA storing. No other storage keys are affected by interrupts.

As well as the above, the NSSC 11 is equipped with a set of short precision (16-bit) instructions which operate with 16-bit fixed point two complement numbers. They are manipulated in the lower half of the general-purpose registers and there is no sign extension as on a 5/360. A long precision fixed point (64-bit) instruction set is available also. An even-odd register pair is used for holding 64-bit numbers and only ADD, SUBTRACT, COMPARE, LOAD and STORE instructions are provided.

IV. ASPS EXECUTIVE FUNCTIONAL DESCRIPTION.

The ASPS executive has the primary goal of providing scheduling in real time of certain processes. There are three time periods of interest. At present, they are 10 milliseconds, 100 milliseconds and 1 second but these may be adjusted. These time periods will be referred to here by the symbols $T(1)$, $T(2)$, and $T(3)$, in millisecond time units. Associated with these time periods are three sets of processes. The set $(P(1,j))$ is associated with $T(1)$, the set $(P(2,j))$ with $T(2)$ and the set $(P(3,j))$ with $T(3)$. Certain computations must be completed every $T(1)$ milliseconds, others every $T(2)$ milliseconds and still others every $T(3)$ milliseconds. $T(2)$ and $T(3)$ are integer multiples of $T(1)$, and $T(3)$ is an integer multiple of $T(2)$.

Timing is centered around an 1/0 interrupt from the DLA which is derived from, but not coincident with, the system master timing pulse. This interrupt will be generated by the DLA every $T(1)$ milliseconds regardless of what the NSSC 11 does, although the NSSC 11 may mask it. At system start-up, the ASPS executive performs any data initializations which are necessary. Signals the DLA that initialization is complete using direct 1/0, and then places the NSSC 11 in the wait state with no processes active.

Processing begins when the first 1/0 interrupt arrives from the DLA. From then on, $P(1,j)$ must be completed every $T(1)$, $P(2,j)$ every $T(2)$ and $P(3,j)$ every $T(3)$ milliseconds of real time for some $j$. Real time can be thought of as a sequence of $T(3)$ time periods. Each $T(3)$ time period is broken into an integral number of $T(2)$ time periods, each $T(2)$ time periods is broken into an integral number of $T(1)$ time periods. The quantity $j$ is called the mode and a different process is used for each mode. The mode is the operating state of the platform and currently $k$ (i.e., the number of modes) is 4. They are called DLA, COURSE, FDL and SL. The system changes mode based on certain inputs (see below) and only certain transitions are valid. Mode changes can only occur at the beginning of a $T(3)$ millisecond time period.

The NSSC 11 interval timer is not used for any determination of real time. It is used solely as a check on the system master timing pulse. At the beginning of each $T(1)$ time period (i.e., following the interrupt from the DLA) the interval timer is loaded with a value slightly larger than $T(1)$. If the timer ever expires then clearly an error has occurred. For the initial version of the executive, if the timer interrupt ever occurs, the system will not attempt to recover but merely inform the operator and enter the wait state.

When the 1/0 interrupt at the beginning of $T(1)$ occurs the executive is entered. The DLA will already have completed a buffered input operation and placed a total of $L(10)$ words into the NSSC 11 memory. $L(10)$ is currently 38. This block of data is in two parts. The first
L(IK)-2 words are data for process P(1,j) and the last 2 are input to another process (see below). Prior to initiating P(1,j), these two words are removed by the executive and used to build a table in a separate memory area.

When P(1,j) completes, a table of outputs of length L(OUT) have been produced. L(OUT) is currently 32. A direct output is sent by the executive to the DEA which then begins a buffered output operation, i.e., the DLA removes the results of P(1,j) for its own use.

After the direct output has been sent, part of P(2,j) is run. For every j, the process P(2,j) must be completed in a T(2) time period. It is organized as a series of subprocesses which, when executed in series, constitute the entire process P(2,j). These subprocesses will be denoted P(2,j,k). For every j and k, the process P(1,j) and P(2,j,k) can be executed sequentially in less than T(1) milliseconds. Clearly k has to be less than or equal to T(2)/T(1) in order to meet the deadline. The breaking of P(2,j) into a series of subprocesses is not a requirement but merely the process structure in the present design.

When P(2,j,k) completes, P(3,j) is resumed. It continues to execute until either:

(a) the next I/O interrupt from the DEA occurs or

(b) P(3,j) completes.

P(3,j) must be completed in a T(3) time period. P(3,j) for the current j is initiated at the beginning of each T(3) time period and following the completion of P(3,j) the processor will be in the wait state. If P(1,j) or P(2,j) are not executing, P(3,j) operates on a table of data which is constructed for execution j of P(3,j) during execution 1-3. The table is L(BACK) words long and is in fact constructed from from the 2 word blocks which were not input to P(1,j) during the executions of P(1,j) (see above). The first of these two words is a key and the second in a data word. If the key is negative, the data word is to be ignored. If the key is positive it consists of two parts. The first is an index indicating where in the data table the data word belongs. The second part is an identifier indicating to which of several possible tables the data word belongs. During any given T(3) time period all of the data words will be intended for the same data table. If the identifier changes during a given T(3) time period, an error has occurred.

Switching of nodes can only occur between execution of P(3,j), i.e., at most only every T(3) milliseconds. One of the constituents of the data table for P(3,j) is a mode change indication. This designates the node which the system will be in for the next T(3) time period. Valid mode transitions have not yet been decided.

In addition to real time management, the executive must respond to the other sources of interrupt on the BSSC II. The machine check and program interrupts are both to be regarded as errors, and processing will consist of informing the operator and putting the system into the wait state. Supervisor call interrupts must provide supervisor services in the normal way and only two such services are presently defined. They are:

(1) SVC code 55(hex) - Process P(3,j) has ended.

(2) SVC code AA(hex) - Process P(1,j) has ended.

External interrupts are to be regarded as errors except for the interrupt generated by the operator from the TSE. Processing in this case is currently undefined and so all external interrupt processing consists of informing the operator and putting the system into the wait state.
V. SYMPATHETIC NERVOUS SYSTEM

Refer to Fig. 3 for the system timing:

(a) The PLA generates a sync pulse every T(1) milliseconds.

(b) Starting at the trailing edge of the sync pulse, the DEA

(c) The DEA interrupts the HSSC II when input is complete.

(d) The HSSC II computes with the data and deposits the output in the

buffer.

(e) The HSSC II then signals the DEA to indicate that data is

available.

(f) The DEA begins to remove data from the output buffer in the HSSC

II.

(g) The HSSC II then performs the next sequential parts of the T(2)

next computation.

(h) Once (g) is complete the HSSC II reverts to background processing.

(i) The sequence repeats.