CSDL-78-010 10/76

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Extensions to, and Implementation of, "A Methodology for Configuring Distributed Real Time Microcomputer Systems with Application to Inertial Navigation Systems"		5. TYPE OF REPORT & PERIOD COVERED 7/1/77-9/31/77
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) J. Wexler, J. Paradiso		8. CONTRACT OR GRANT NUMBER(s) F33615-75-C-1149
9. PERFORMING ORGANIZATION NAME AND ADDRESS The Charles Stark Draper Laboratory, Inc. 555 Technology Square Cambridge, MA 02139		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS Task 4.2.1
11. CONTROLLING OFFICE NAME AND ADDRESS Air Force Avionics Laboratory Wright-Patterson Air Force Base Dayton, Ohio 45433		12. REPORT DATE October 1976
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

P2653

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Real Time Computer Systems, Mixed Integer Linear Programming, Mass Memory Systems, 1553A Data Bus

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report is an extension of the work performed in "A Methodology for Configuring Distributed Real-Time Microcomputer Systems with Application to Inertial Navigation Systems." That report developed a methodology for characterizing, in a mathematical formulation, the goals and constraints involved in configuring such a system.

7512F376

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ACKNOWLEDGEMENT

This report was prepared under USAF Contract F33615-75-C-1149 by Charles Stark Draper Laboratory, Inc., Cambridge, Ma. The monitoring Air Force project engineer is Captain E. Harrington (RWA-2), Air Force Avionics Laboratory, Dayton, Ohio.

Dr. Gary Kochman of Bell Laboratories provided the starting point for the implementation of the algorithm with his computer program code "BB ".

The authors wish to express their appreciation to Mr. William Shepard and Mr. Dave Kaiser of the Air Force Avionics Laboratory for their assistance during the course of this contract.

FORWARD

This report is a follow-on effort for "A Methodology for Configuring Distributed Real-Time Microcomputer Systems with Application to Inertial Navigation Systems," which was also prepared under the sponsorship of the Air Force Avionics Laboratory, Dayton, Ohio, under USAF Contract F33615-75-C-1149.

In this report, the following is presented:

Part I -

1) An expansion of the conceptual model (of distributed real-time systems) to include a) the 1553A data bus structure and b) mass memory considerations.

Part II -

2) The method of, and results from, implementing a computer program which performs the 'optimization' required to configure a real-time system. This computer program makes use of branch-and-bound mixed integer linear programming techniques.

Part I

Expanded Model

Table of Contents

I) The 1553A

The 1553A is a military standard specification for Data Bus operation accepted by the Defense Dept. 1553A-related busses have been applied in the Bl, F-18, and space-shuttle architectures.

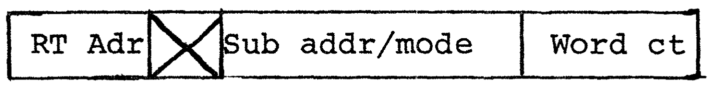
There are two basic varieties of devices appended to the 1553A bus: Controllers (CT) and Remote Terminals (RT). Controllers monitor and maintain the bus operation by initiating all communication processes and responding to error conditions. Remote Terminals are "slaves" which remain inert until otherwise directed by a Controller. Occasionally provisions are made for a remote terminal to interrupt a controller via external connections. In small-scale systems, Controller functions may be realized within a "shared" processor which is also responsible to other tasks. In larger systems, however, the controller may require totally dedicated hardware.

The "bus" itself is generally a shielded twisted pair. Devices are patched to the bus via decoupling transformers ("stubs"). Manchester II data encoding is employed, and the transmission rate is specified as 1 Mhz.

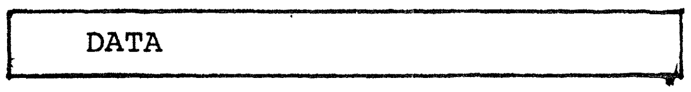
Data is transmitted in 20 bit words. The first three bits are dedicated to a synchronization signal, and the last bit is used for parity, leaving us 16 bits free for data transferral.

There are three types of transactions which occur on the bus.

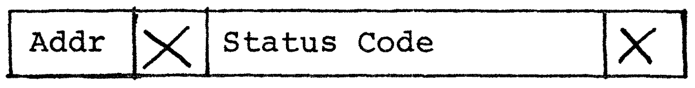
I) Command words: Sent from a controller to a RT.
Format......



II) Data word....sent by all.



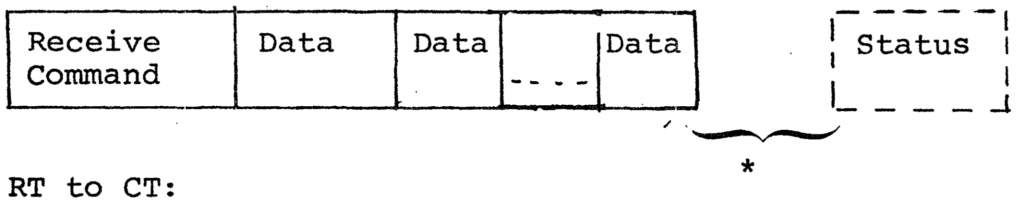
III) Status Word: Sent from RT to Controller



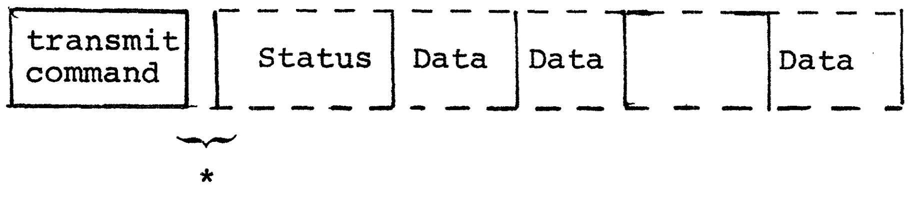
Using this language, three types of tranactions are possible:

—— = CT
--- = RT
*= 2-5μsec delay

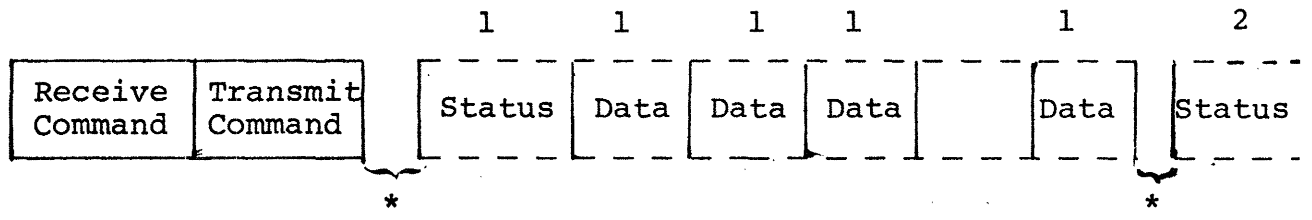
I) CT to RT:



II) RT to CT:

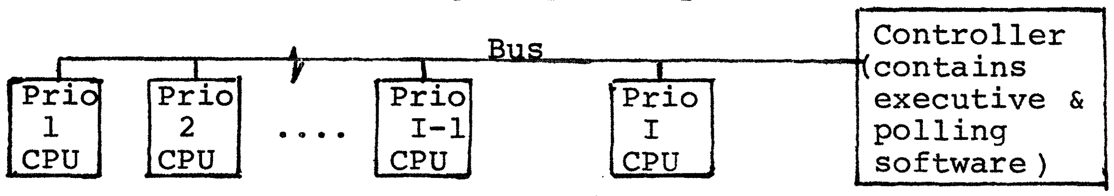


III) RT_1 to RT_2

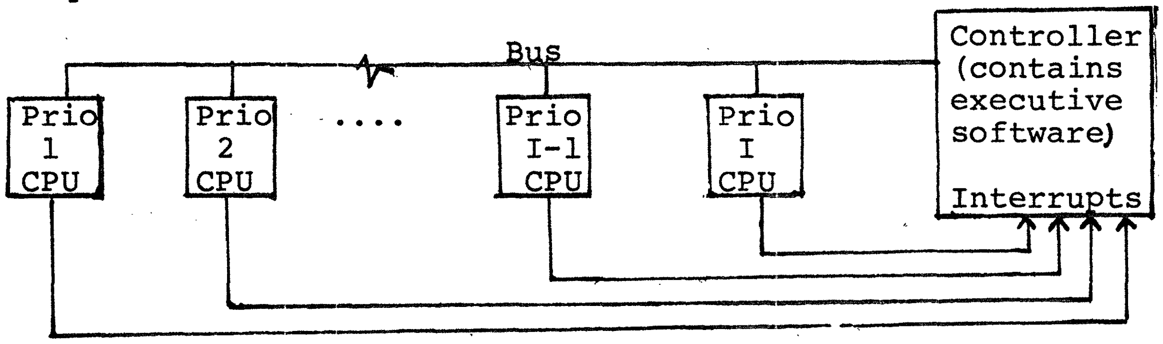


RT's don't respond until prompted via a XMIT command from the Controller. Thus, in this type of single-bus controller/slave system, the controller must poll the RT's in order to initiate data transferral (barring the external interrupt connection).

In a multi-processor configuration, the active processors sitting on a bus must possess controller attributes. If they were RT's, they would be incapable of initiating bus activity, necessitating a polling scheme of sorts. This multi-controller bus structure suggests a priority hierarchy whereby "crucial" processors may interrupt a lower-priority communication, and seize control of the bus. No mention of bus "priority" has been found in the 1553 literature, and it seems to fall outside the realm of the Military Standard Guidelines. For the purpose of modeling, however, we may assume the existence of a prioritized structure, and proceed with the analysis. <u>Note:</u> There are several schemes by which each processor may be assigned a unique priority level on the 1553A. In a single controller polling system all executive software will reside in the controller itself, and other CPU's will act as Remote Terminals. The controller will poll all CPU's for I/O requests over regular intervals, and grant bus privileges first to processors with higher priority.



Another configuration may be used whereby the remote CPU's interrupt the Controller via an external connection. In this case the polling function of the controller is no longer required.



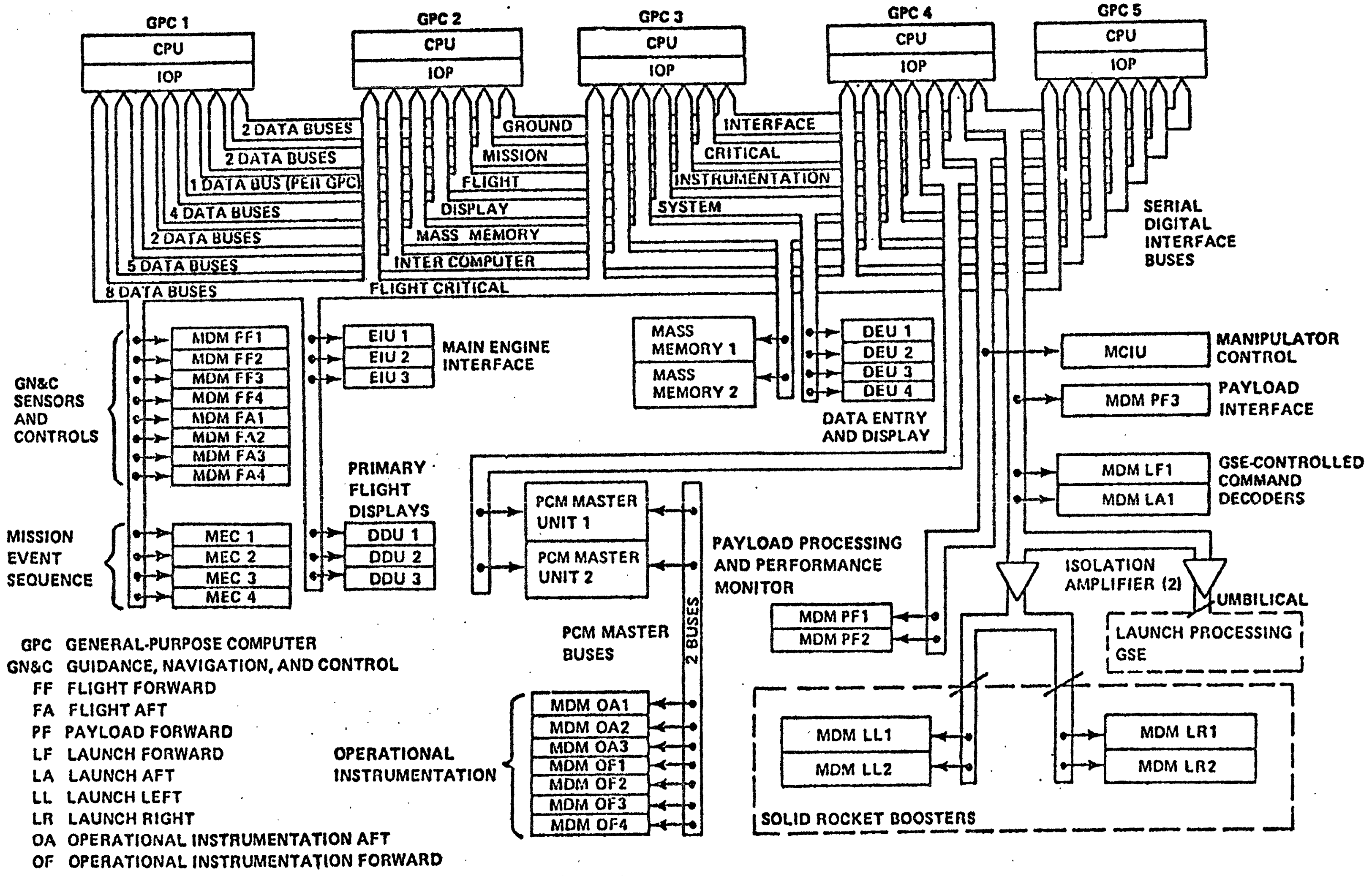
Both of these methods may be combined into a hybrid, where high priority, time critical CPU's may interrupt the controller externally, and lower-priority CPU's are polled by the controller over the bus.

The controller is depicted as being an additional processor in the above diagrams, but this is not necessarily ture; its functions may be incorporated into one of the CPU's.

Another feature found in the application of the 1553A is redundancy, where two or more bus systems link processors and peripherals. This duplication serves a dual purpose:

1. In the event of a hardware failure on Bus A, all communications can be diverted to Bus B, thus preserving system integrity.
2. Bus communication may be evenly split between Busses A and B, lowering the loading conditions and cutting system overhead.

Redundancy is not restricted to a two-bus system; multiple bus structures seem common practice in avionics. A linear programming scheme could be applied to optimize redundant bus utilization, whereby both bus traffic and re-configuration in the event of a bus failure may be determined. This problem is an entity in itself, however, so the discussion here will be limited to single bus systems. [A diagram depicting bus utilization in the space shuttle is included to illustrate the level of complexity associated with multiple-bus systems.   The busses



Shuttle Orbiter data processing subsystem.

in the space shuttle are shared between five processors and supervised by Bus Control Elements (BCE's) which are directed by a Master Sequence Controller (MSC). The shuttle also contains two tape drives as Mass Memory Units (MMU's). The mass storage is used to retain non-time critical data and IPL programs. The worst-case access time for the shuttle's MMU is 60 seconds].

II) Mass Memory

Mass memory is frequently used in spacecraft, and has been employed in satellites since the mid 1960's. The type of memory has generally been magnetic tape*, and it has had several applications:

I) Buffering of data for telemetry (read/write storage require

II) Storage of IPL routines (read only)

III) Program storage (displays, etc) [Usually read only]

IV) Misc. data storage (read/write).

Magnetic tape devices can present difficulty due to wear on the mechanical,moving portions of the apparatus. Work is underway to replace mag-tape with smaller, more dependable methods such as magnetic domain ("bubble") memories, electron beam memories, laser written optical memories, etc. These schemes promise imminent success, but as of the present moment, magnetic tape is most frequently applied.

The application of a tape-drive unit as mass memory limits

*Floppy disks are beginning to be used in modern aircraft for semi-time critical applications such as digital map storage,etc. The mean access time entailed in these random-access devices present a large improvement over tape driven systems but still lags sufficiently behind actual semi-conductor memories to question the validity of a ROM/mass memory tradeoff.

us to sequential access, with the data generally structured in conventional file-record format. The memory unit will accept three classes of commands; read, write, and position. A special write-protect command is sometimes included.

Mass storage of this sort can not be used as rapid access memory. The application must be limited to the scope of the above list, i.e., as a large store of non-time critical programs and data. The write function usually accomplishes two purposes:

1. Temporary data storage for telemetry and on-board processing.
2. Capacity to alter programs if desired.

The first item above is irrelevant when applied to the topic of avionics control. The second item holds much potential, and should be carefully considered when structuring a system, but it is insignificant with respect to our modeling scheme since the slow access times prevent the device from being used as an effective RAM. For the purposes of this study, then, I believe that we can consider these types of mass memory units (MMU's) to be akin to very slow ROM's.

The MMU's may be connected to the rest of the system via an interface to a 1553A type bus as remote terminals. In the space shuttles, there are two MMU's for redundancy and overhead-splitting purposes.

The mass memory concept outlined here introduces no new program variables. Program constants are specified which dictate the mass-memory requirements of various tasks, and the delay times entailed. The only impact of mass memory upon our scheme,then, is a fixed bias added into the cost, volume, mass, power, and timing constraints. Nothing dealing with the MMU is optimized, primarily because of the nature of the devices now applied. Assuming the introduction of smaller, faster hardware such as electron beam memories, mass storage could compete with ROM, providing us with program control over the MMU/ROM trade-off. Another possible program variable is determined by the position of particular files on a sequential device (such as tape). Assuming the position to be proportional to access time, the order of files could be optimized such that time-critical tasks receive data with minimal wait.

III) An Attempt at a Modeling Strategy

In this section, the modeling postulates proposed in Chapter Six of J. Wexler's "A Methodology for Configuring Distributed Real-Time Microcomputer Systems with Applications to Inertial Navigation Systems" will be modified to suit the previously described devices. All equations referenced come from that test.
(Note: In this section it is assumed that there are I processors and J tasks).

Equation 1M, which calculates the expense of the system, will require an extra term specifying the cost of the MMU's [it is assumed that all interfacing hardware for the 1553A bus is included in the CPU cost, "CPUC"]. Thus 1M becomes:

$$COST = CPUC \bullet I + PWRC \bullet P + RAMC \sum_i RAM_i + ROMC \sum_i ROM_i + MMUC \bullet M$$

(It is assumed that all MMU's are shared among processors).

The weight constraint will, of course, be affected by the addition of a MMU: (equation 3-M)

$$CPUW \bullet I + PWRW \bullet P + RAMW \sum_i RAM_i + ROMW \sum_i ROM_i + MMUW * M \leq CNSW$$

And so the volume constraint: (equation 4-M)

$$CPUV \bullet I + PWRV \bullet P + RAMV \sum_i RAM_i + ROMV \sum_i ROM_i + MMUV * M \leq CNSV$$

And thus the Power constraint: (equation 5-M)

$$CPUP \bullet I + PWRP \bullet P + RAMP \sum_i RAM_i + ROMP \sum_i ROM_i + MMUP \bullet M \leq 0$$

Where:	MMUW	= weight of a single MMU
	MMUC	= Cost of a single MMU
	MMUV	= Volume of a single MMU
	MMUP	= Power requirement of a single MMU
	M	= Number of MMU's on the system

(all of the above are problem constants)

It is assumed that the weight, volume and power contributions from the 1553A apparatus is contained in the CPU terms.

Equations 10-M and 11-M, which regulate the amount of data transferred over the bus during a major cycle must also be modified to include the addition of mass memory traffic:

$$\sum_j \sum_n (1-BOTH_{nj}) \bullet DATA_{jn}) + MEMQ \sum_j MENT_j \leq BUSL \qquad (10\text{-}M)$$

$$\sum_j (TA_{ij} \sum_n ((1-BOTH_{nj}) \bullet (DATA_{nj}+DATA_{jn}))+MEMQ \bullet MENT_j) \leq TERL \qquad (11\text{-}M)$$

$$i=1,\ldots I$$

where: MEMQ = Number of data words per block of MMU storage

$MENT_j$ = Number of MMU storage blocks required by task j

(All of the above are problem constants)

The next equation which requires extensive modification in order to incorporate the 1553A/mass-storage models is 16-S. In the thesis, (16-S) takes the form:

$$TERM_j=INIT_j+EXEC_j+I/O_j+(EXEC+I/O)_{>j}$$

This equation calculates the termination time of a particular task from its initiation,execution, and I/O time requirements (adding the execution and I/O time of all tasks residing in the same CPU which interrupt task j). Our bussing system and Mass Memory devices will add terms to this equation:

$$TERM_j=INIT_j+EXEC_j+I/O_j+(EXEC+I/O+MMUTIME)_{>j} + MMUTIME_j+BUSTIME_j$$

$MMUTIME_j$ represents an access delay for the mass memory unit. For a random access device, this term is a program constant. In sequential configurations, we may be able to take MMUTIME to be a "worst case" delay, thus keeping it constant. MMUTIME will then be in the form:

$$MMUTIME_j = TIMM*MEMT_j$$

Where: TIMM = Time (% of major cycle) needed to access a block of data [worst case].

$MEMT_j$ = Number of data blocks required by task j.

The MMUTIME term can be specialized to fit a certain configuration if desired. For instance, assume that our MMU is a tape drive which always rewinds to the load point after being used by a task (take the rewind time to be negligible). We can then define the following problem constants:

SKIPTIME - Time required to "skip" a block of data (without reading).

TIMM - Time required to read a block of data

$MEMT_j$ - Number of blocks required for task j.

We can then introduce the following program variable:

$$\left.\begin{array}{l} [PLACE_{i,j}] \\ i = 1,2,\ldots J \\ j = 1,2,\ldots J \end{array}\right\} \begin{array}{l} = 1 \text{ if the file used by task i is located sequentially after the file used by task j.} \\ = 0 \text{ otherwise} \end{array}$$

Constraints must be held on PLACE which prevent more than one task from occupying the same position on the tape. (It is possible that a file may be shared among tasks, in which case these constraints must be modified and/or new program constants introduced). Thus $\text{PLACE}_{i,j}$ is defined over all i, j such that

$$\text{MEMT}_j \bullet \text{MEMT}_i \neq 0 \ \underline{\text{and}}\ \text{PLACE}_{i,j} + \text{PLACE}_{j,i} = 1$$

The MMUTIME term will be:

$$\text{MMUTIME}_j = \text{SKIPTIME} \bullet \sum_{k=1}^{J} \text{PLACE}_{jK} \bullet \text{MEMT}_K + \text{TIMM} \bullet \text{MEMT}_j$$

By optimizing PLACE, we obtain the best order of files on the tape.

The BUSTIME_j term represents the delay task j experiences in waiting to use the bus. If one assumes a non-prioritized polling type bus, BUSTIME_j will be a problem constant which specifies a "worst case" wait interval.

However, if we introduce a priority hierarchy on the bus, the entire scheme changes. We'll assume an array of I processors, each of which occupies a unique level on the bus.

Let us define a matrix as problem constant:

$$[\text{BPRI}_{i,j}] = \begin{cases} 1 \text{ if processor } i \text{ has higher bus priority than processor } j \\ 0 \text{ otherwise} \end{cases}$$

i=1,2,...I
j=1,2,...I

Another square matrix may be derived from this one:

$$[\text{BTPR}_{ij}] = \begin{cases} 1 \text{ if task } i \text{ resides in a CPU with higher bus priority than task } j \\ 0 \text{ otherwise} \end{cases}$$

i=1,2,...J
j=1,2,...J

BTPR is easily formed via a linear transformation through the task assignment matrix:

$$[BTPR] = [TA]^T [BPRI]\ [TA]$$

We can define a timing constant:

TIMB = Length of time (% major cycle) for one unit (word) of information to be transmitted over the bus.

Thus we can define $BUSTIME_j$ to be the worst-case delay which occurs whenever all tasks with bus priority higher than task j grab the bus simultaneously. (The matrix $BUPT_{kj}$ is defined as unity whenever Task K has the potential to actually interrupt $task_j$ on the bus. BUPT is formed by holding BTPR to the constraints listed below.)

$$BUSTIME_j = \sum_k BUPT_{kj} \left(\sum_n \left((1-BOTH_{nk}) \cdot TIMB(DATA_{nj}+DATA_{jn})\right)\right)$$

It is assumed that 1553A command and status words are included as DATA. The higher CPU priority term in the equation for $TERM_j$ (term with ">j" subscript) requires no additions, since all interrupting tasks reside in the same CPU as task j, and have the same bus priority (every task with high bus priority is included in $BUSTIME_j$ as it now stands).

We can relax somewhat from the "worst case" world modeled above by adopting additional constraints on task timing. Since all tasks do not "overlap" temporally, they do not always request simultaneous bus usage. Task phasing is taken into account via the constraints listed in figure 8-2 (page 165) of J. Wexler's thesis. Three modifications will be necessary:

I. We will have to open the scope over which equation 17-S is implemented. It will hold for all n, j such that $BOTH_{nj}=0$ and $DDEP_{nj}=DDEP_{jn}=0$ (since potential bus conflicts arise only between any pair of tasks in different processors which are not "data dependent).

II. We will have to introduce a new constraint equation analogous to equation 20-S:

$$BUPT_{nj}-.5\ BEFT_{nj}-.5\ BEFT_{jn} \leq 0 \qquad (24\text{-S})$$

Scope is over all n, j such that $BTPR_{nj} = 1$ and $BOTH_{nj} - 0 = DDEP_{nj} = DDEP_{jn}$

III. We'll have to introduce yet another constraint equation, analogous to 21-S:

$$BEFT_{nj}+BEFT_{jn}-BUPT_{nj} \leq 1 \qquad (25\text{-S})$$

Scope is over all n, j such that $BTPR_{nj}=1$ and $BOTH_{nj} = 0 = DDEP_{nj} = DDEP_{jn}$

If we adopt both the sequential mass memory, and prioritized bus systems discussed above, equation 16-S will now read:

$$\begin{aligned} TERM_j = {} & INIT_j+EXEC_j+\sum_n((1-BOTH_{nj})\bullet(TIMI\bullet DATA_{nj}+TIMO\bullet DATA_{jn})) \\ & +\sum_k(BUPT_{kj}\sum_n((1-BOTH_{nk})\bullet TIMB\bullet(DATA_{nj}+DATA_{jn}))) \\ & +\sum_k(RUPT_{kj}\sum_n((1-BOTH_{nk})\bullet(TIMI\bullet DATA_{nj}+TIMO\bullet DATA_{jn}) \\ & \qquad +SKIPTIME\bullet PLACE_{kn}\bullet MEMT_n)+TIMM\bullet MEMT_k) \\ & +SKIPTIME\bullet\sum_k PLACE_{jk}\bullet MEMT_k + TIMM\bullet MEMT_j \end{aligned}$$

All newly defined variables were described earlier.

IV) Conclusions

During the course of this article, the 1553A data bus and mass memory apparatus deployed in avionics systems have been discussed. An attempt at modeling this hardware into the methodology proposed by J. Wexler in his thesis (CSDL T-646, reference #1), indicated that simplified MMU storage and I/O polling busses merely appear as additive program constants in the timing and sizing constraints. New program variables were introduced via a sequential MMU system and a priority driven bus protocol, however, Modifications to several constraint equations were suggested to account for these additions.

IV) References:

1) Wexler, J.W.,"A Methodology for Configuring Distributed Real-Time Microcomputer Systems with Application to Inertial Navigation Systems", CSDL#T-646, June 1977.

2) MIL-STD-1553A (USAF), 30 April 1977.

3) Boose, E.F., "Lessons Learned Through a MIL-STD-1553 Time Division Multiplex Bus", pgs. 3-18 of "RIDS Papers for Naecon '75" , MITRE Corp. Technical report, number MTR-3023, 12 May 1975.

4) "Prime Item Development Specification for DAIS Multiplex Remote Terminal", DAIS Spec. No. SA301301, 14 April 1975.

5) "Prime Item Development Specification for DAIS Digital Command/Response, Time Division Multiplexing Data Bus", DAIS Spec. No. SA301302, 14 May 1975.

6) "Addendum Specification for DAIS Bus Control Interface Unit Operations and Design Description", DAIS Spec. No. SA301400, 12 May 1975.

7) "Space Shuttle Program, Orbiter Project, Computer Program Development Specification, Volume I, Book I", Chapter 3.3, "Mass Memory Unit (MMU)", No. SS-P-0002-110, February 10, 1975.

8) Curtis, D.A. "A Study of Mass Memory Applications", Arthur D. Little,Inc., No. NASA-CR-116242, August 1970.

9) Kasmala, A.L. "Engineering Study for a Mass Memory System for Advanced Spacecrafts, Final Report", Intermetrics,Inc. No. NASA-CR-108672, July 1, 1970.