implementation of this solution has been described, with most attention devoted to reliability problems.

At this preliminary stage of design, the main parts of a first prototype have been built. These include a single-cylinder engine installed on a test bench, with suitable measurement instrumentation, as well as the microcomputer system and interfaces. A modeling and simulation of the four-bar mechanism and optimal valve-timing adjustments have been performed. A final assembly based on these results is expected to be operational shortly.

REFERENCES

[1] K. Ikeura, A. Hosaka, and T. Yano, "Microprocessor control brings about better fuel economy with good drivability," Society of Automotive Engineers, paper no. 800056.
[2] T. Toyoda, T. Inoue, T. Yaegashi, and K. Acki, "Electronic engine control systems for the smaller passenger car," Society of Automotive Engineers, paper no. 800894.
[3] *UPI41 User's Manual*, Intel, Santa Clara, CA.
[4] *8085 User's Manual*, Intel, Santa Clara, CA.
[5] A. D. Toelle, "Microprocessor control of the automobile engine," Society of Automotive Engineers, paper no. 770008.
[6] A. Avizienis, "Fault tolerance: The survival attribute of digital systems," *Proc. IEEE*, vol. 66, no. 10, Oct. 1978.

A Real-Time Engine Simulator Using Multiple Microcomputers

JUAN R. PIMENTEL, MEMBER, IEEE, AND MICHAEL T. LOEFFLER, MEMBER, IEEE

Abstract—**A multiple microcomputer system which is useful for real-time simulation of both complex and fast response systems has been designed and constructed. Interprocessor communication between five microprocessors is done in parallel using direct memory access techniques. Tests on system performance indicate that it is capable of fast processing element execution times and fast interprocessor message transfers.**

The system was used to simulate a spark-ignition internal combustion engine in real time. The simulation provides manifold pressure, net torque, engine speed in response to throttle angle, spark advance, exhaust gas recirculation, and load torque inputs. The engine simulation model was decomposed into four sections: carburetor, intake manifold, combustion, and dynamics. Potential applications of the engine simulator include the development of control algorithms and engine designs.

I. INTRODUCTION

SYSTEM SIMULATION has been useful in engineering design for various reasons: first, the designer might want to know how the designed system behaves in different situations [1]; and second, simulation is used as a tool for the design process [2]-[4]. Real-time simulation consists of obtaining the simulated variables in the same time frame that the actual variables evolve in the system being simulated. The relationship among all variables involved in the simulation is given by a model for the system. There are basically three approaches used for real-time simulation: analog, digital, and hybrid (a combination of analog and digital).

Latest developments in microprocessor and microcomputer technologies add a new dimension to real-time digital simulation. The advances have been so phenomenal that it is now possible to buy an integrated circuit that has a central processing unit, enough memory to hold most practical programs, analog to digital converter, timers, and input/output (I/O) ports. These circuits, called microcomputers, are revolutionizing design concepts is several engineering areas. Moreover, microcomputers are designed in such a way that they can be interconnected in any array configuration (multiple microcomputers) to perform more difficult tasks.

If one could decompose the system under simulation into functional units such that each unit could be simulated by one microcomputer, then it appears that multiple microcomputers are very promising for real-time simulation. The main advantage of using multiple microcomputers is that they increase system throughput. In simulation they can be used to decrease processing time and achieve real-time simulation. The object of this paper is to present a multiple microcomputer hardware configuration for real-time engine simulation. The simulator, which consists of an array of five microcomputers, simulates a spark-ignited, internal combustion engine.

II. SIMULATION OF AN INTERNAL COMBUSTION ENGINE

Automotive vehicle simulation is useful in several stages of the automobile process. Gilmore [3] considered an analog simulation for a hybrid gasoline-electric vehicle in order to study the effects of engine size, electric motor size, flywheel

Manuscript received June 1, 1982; revised December 22, 1982.

J. R. Pimentel is with the Department of Electrical and Computer Engineering, GMI Engineering and Management Institute (formerly General Motors Institute, General Motors Corporation), Flint, MI 48502.

M. T. Loeffler is with Pontiac Motor Division, General Motors Corporation, Pontiac, MI 48053.

0278-0046/83/0500-0117$01.00

TABLE I
SUMMARY OF AUTOMOBILE ENGINE MODELS

METHOD	FEATURES	AUTHOR
Regression Techniques	Applied to Data Obtained from Automated Dynamometer Tests	Tennant
Process Identification	Model Valid Only for Certain Operating Conditions	Morris
Taylor Series Expansion	Model Valid for Certain Operating Conditions	Cassidy
Table Look-Up Based on Engine Data	Model Valid for All Operating Conditions	Dobner

size, and battery pack requirements on overall vehicle performance. Mitchell *et al.* [2] considered a digital simulation of vehicle dynamics to driver inputs such as steering or braking, and to external disturbances such as wind gusts. Reid and Graf [5] considered linear models to represent driver response during an obstacle avoidance maneuver. Kotwicki [6] considered a dynamic model for a torque converter to allow mathematical analysis and performance prediction of powertrain controls.

In the last few years, automobile manufacturers are using electronic controllers to control certain variables such as air-fuel ratio, idle speed, cruise speed, etc., in order to minimize fuel consumption and emissions. Dynamic engine models that would allow one to design or evaluate engine controllers are important because they would reduce controller design time and cost. Tennant *et al.* [7] considered a model that uses regression techniques on data obtained from automated dynamometer tests. Morris *et al.* [8] considered a process identification approach to model an engine at a given operating condition. Cassidy *et al.* [9] used a linearization method to obtain a model valid also for a given operating condition. Dobner [10] and Coats and Fruechte [11] attempted to characterize an engine by a nonlinear model useful at all operating conditions. Table I summarizes the major models for an automobile engine.

III. ENGINE MODEL

Dobner's model has been chosen for the real-time simulation because it is a global model as opposed to the incremental one used by Cassidy. The model accepts the following variables as inputs: throttle position (α), spark advance (S), air-fuel ratio (A/F), exhaust gas recirculation (EGR), and load torque (TL). The output variables are: engine speed (N), net torque (TN), intake manifold pressure (PM), exhaust manifold pressure (PE), and friction torque (TF). The model gives functional relationships for the output variables in terms of the input variables. The basic approach of the model is to take into account available engine data obtained from dynamometer tests for a specific engine and to model the dynamics separately. The dynamics are given by time delays and integrations in the "intake manifold", "combustion", and "dynamics" sections, respectively.

Due to the complex (highly nonlinear) relationships among variables, a brief description of the model will be described using arbitrary functions which will give a certain variable in terms of other variables shown explicitly in the relationships.

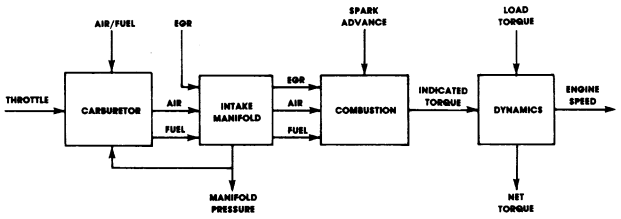

Fig. 1. Basic model configuration for an internal combustion engine.

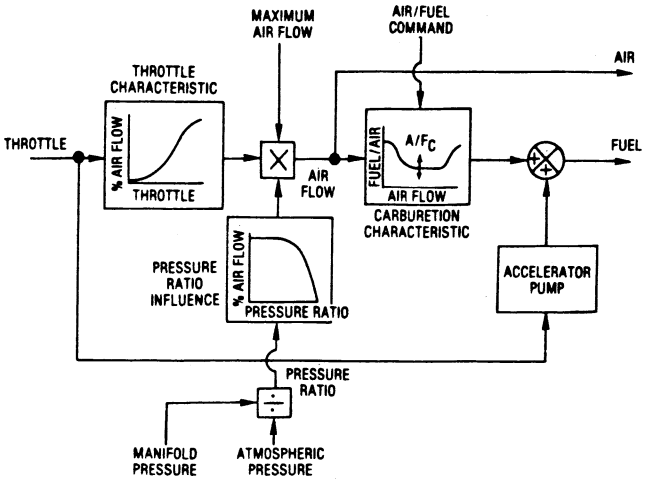

Fig. 2. Carburetor model.

Furthermore, some variables have to meet certain constraints which will be stated after the relationships. For example, manifold pressure can never be greater than atmospheric pressure. The engine characteristics are: 8-cylinder, 7 l, bore = 4.25 in, stroke = 3.76 in, manifold temperature = 110°F. Fig. 1 shows the basic model configuration which was motivated by the physical arrangement of the engine. The input and output of each block is a variable that is measurable in the laboratory. Each section is briefly described next.

A. Carburetor

The carburetor provides the air and fuel flow into the intake manifold in response to the accelerator throttle position. Air flow is a function of throttle angle α, and manifold pressure PM. Fuel flow is mainly a result of the carburetor characteristic (except for power enrichment and accelerator pump). Functional relationships for the carburetor model are summarized as

$$AR = f_1[\alpha, PM] \tag{1}$$

$$FR = f_2[AR] \text{ Carburetion characteristic.} \tag{2}$$

Where AR is the mass air rate and FR is the mass fuel rate. Fig. 2 shows the carburetor model in detail.

B. Intake Manifold

This section models the air, fuel, and EGR rates into the cylinders. The carburetor and EGR valve controls the flow into the manifold, while the demands of the cylinders dictate the flow out of the manifold. The flow through the manifold is treated with transport delays, and the mass of each constituent is calculated by integrating the difference between the flow in and the flow out. Relationships for the intake

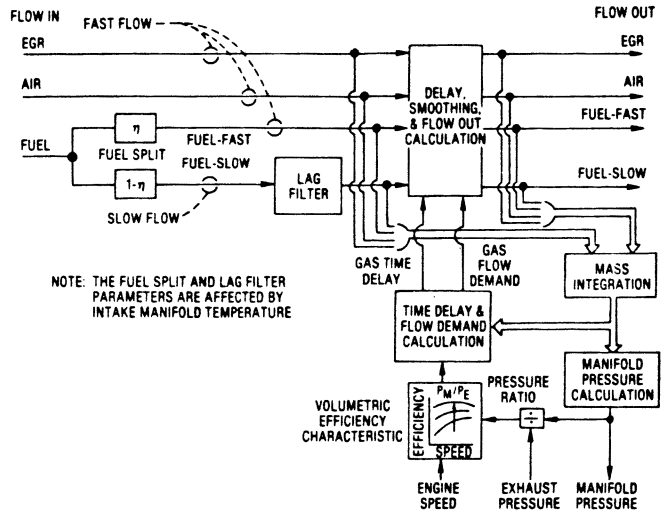

Fig. 4. Combustion model.

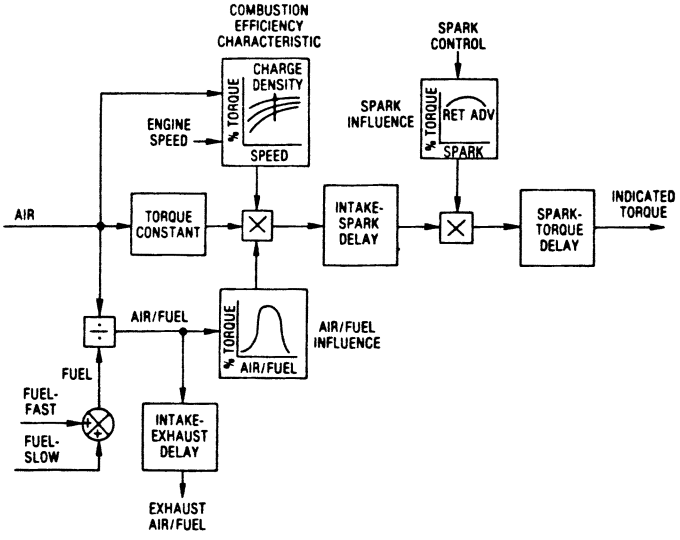

Fig. 3. Intake manifold model.

manifold are summarized as

$$ARO = f_3[MA, N] \tag{3}$$

$$FRO = f_4[MF, N] \tag{4}$$

$$ERO = f_5[MEGR, N] \tag{5}$$

$$MA = \int (AR - ARO)\, dt \tag{6}$$

$$MF = \int (FR - FRO)\, dt \tag{7}$$

$$MEGR = \int (ER - ERO)\, dt \tag{8}$$

$$PM = f_6[MA, MEGR]\,;\ PM \leqslant 14.7 \text{ psi} \tag{9}$$

where ARO, FRO, and ERO are the air rate, fuel rate, and EGR rate out of the manifold. MA, MF, and $MEGR$ are the air, fuel, and EGR masses in the manifold. Fig. 3 shows further details of the intake manifold mode.

C. Combustion

This block generates torque in relation to the air injected into the cylinder. The torque produced is influence by air-fuel, efficiency of the engine, and spark advance angle. There is a time delay that accounts for the fact that torque is produced a finite time after the charge is injected into the cylinder. The model employs a fixed crank angle delay of 270° to provide a time delay which is inversely proportional to engine speed. The indicated torque is expressed as

$$TI = F_7[A/F, S, ARO, N]\,. \tag{10}$$

Further details are shown in Fig. 4.

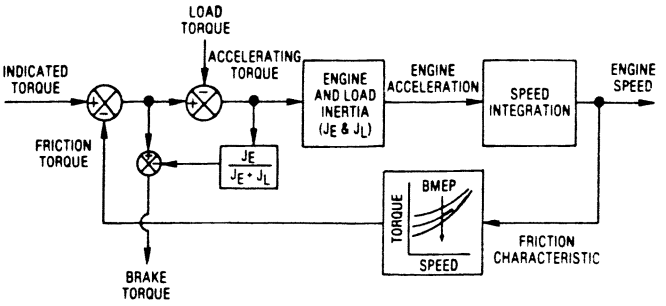

Fig. 5. Dynamics model.

D. Dynamics

This block provides engine speed based on indicated torque. The equations are

$$TF = f_8[N, TB] \tag{11}$$

$$TB = TN - TF \tag{12}$$

$$TN = TB - TA \tag{13}$$

$$TA = f_9[N] \tag{14}$$

$$N = \frac{1}{J_T} \int (TN - TL)\, dt;\ 500 \text{ r/min} \leqslant N \leqslant 5000 \text{ r/min} \tag{15}$$

where TF, TB, TN, and TA are friction, brake, net, and accessory torque. N is engine speed and J_T is the combined engine and load inertia. Fig. 5 shows further details of the dynamics model.

IV. MULTIPLE MICROCOMPUTERS AND SYSTEM DESCRIPTION

The area of multiple microcomputers has been an intensive research area in the last few years [12]-[15]. However, most contributions are theoretical in nature which deal with suggested architectures, communication mechanisms, and implementations. Actual multi-microcomputer implementations are few. Consequently, much experimental work is required to gain experience with the numerous alternatives available.

The nomenclature used will be the following. A microcomputer consists of an interconnected system of microprocessor, memory, and input/output devices. A multiple microcomputer is an interconnected system of microcomputers. It will be assumed that any microcomputer can communicate with any other through a system interconnection mechanism. In the context of multi-microcomputers, a microcomputer is also

called a processing element (PE). A path is a medium by which a message is transferred between the system elements. A switching element is an intervening intelligent entity between the sender and the receiver of a message. A PE processing time is the time required to execute the program associated with a PE.

The following is a discussion of multiple microcomputers as they relate to the simulation implementation of the engine model.

Application of a multiple microcomputer system requires that the problem can be broken down into units that are each assigned to a single PE. The design of the system using a multiprocessor approach brings about several problems:

a) functional decomposition of the problem;
b) task allocation among processors;
c) processor interconnection structures;
d) control of system resources;
e) software structure; and
f) deadlock avoidance.

The following discussion will treat each of the above problems in the context of the design and development of the engine simulator.

A. Functional Decomposition

Several decomposition schemes have been suggested in the literature [16]. Most schemes attempt to minimize a performance measure to obtain optimal workload partitions over a network configuration. Buckles and Hardin [17] use mathematical programming methods to minimize costs for accessing data, storing data and functions, running modules on PE's, and interprocessor transfer of control. Stone [18] approached the problem using graph theory. He considered two cost functions: the cost of interprocessor communications, and the computational cost of a program. Gylys and Edwards [19] considered an heuristic method to minimize the interprocessor message transfers subject to constraints in the number of PE's, memory size of each PE, and PE execution time.

A heuristic method for system partitioning will be considered in this paper. This method works well for situations in which the amount of interprocessor communication is not high. The method is based on the fact that the blocks that result from the partition of the overall system resemble the physical process of the model. The main advantage of this is that the processing blocks are associated with physical sections, and model implementation and verification can be easily done. In addition, variables used in the interprocessor communication have physical meaning, which can aid in the understanding and interpretation of the model. Because the partitioned system resembles the physical system of the model, the disadvantage of the method is that it is problem dependent. However, the advantages of the method far outweigh this disadvantage. The physical partitioning method is a top-down approach applied to modeling. Consequently, it has all the advantages of top-down techniques as used in software development. These advantages are as follows.

a) The partioning is easy to make initially, since it follows the physical arrangement of the system.

b) It is easy to change as the modeled system changes. This feature makes it a highly modular system that could be adapted for other systems as well.

c) Program design is simplified. This is a direct consequence of the top-down approach used.

d) Program coding is straightforward. It is easier to code the small modules that result from the decomposition.

e) Top-down techniques for testing and debugging can be used. For example, we can use "program stubs" to test the program that does the message interchange.

Dobner's engine model [20] is a table-lookup oriented model that is able to describe the response of an internal combustion engine to the following set of dynamically varying inputs:

a) throttle angle;
b) load torque;
c) spark advance;
d) commanded air/fuel ratio; and
e) exhaust gas recirculation (EGR).

The model calculates the dynamic responses to these inputs and presents the following outputs:

a) engine speed (RPM);
b) manifold pressure; and
c) net torque.

The engine model lends itself to the heuristic system partition method described earlier. From Fig. 1 one can see that the functional units are: fuel delivery (carburetion or fuel injection), intake manifold dynamics, combustion characteristics, and mechanical dynamics. Note the few variables that these functional units need to share. These characteristics enable a simpler system architecture and result in a message intercommunication mechanism occupying a small fraction of the overall system computation time.

B. Task Allocation

Task allocation refers to the actual assignment of the functional blocks discussed above to a given PE or a group of PE's. The assignment of blocks depends primarily on the real-time constraints placed upon the system, dictating the number of calculations that each PE is able to perform. Some blocks may be further subdivided or combined to even the processing time required by each PE. For the engine simulation, the computational loads are roughly equal between the four functional units, with the exception of the fuel delivery section. This section is the simplest and requires much less processing time. The effects of this arrangement will be discussed later.

C. Multiple Microcomputer Architecture

Multiprocessor architecture has been studied extensively as it applies to distributed computing systems [21]. The engine simulation application is real time in nature, limiting the possibilities for system architecture. The control structure of the multiprocessor configuration used must be simple, so that the time constraints may be met without wasting processing time on complex control schemes.

The system architecture chosen consists of a central processing element and a number of peripheral PE's connected to a

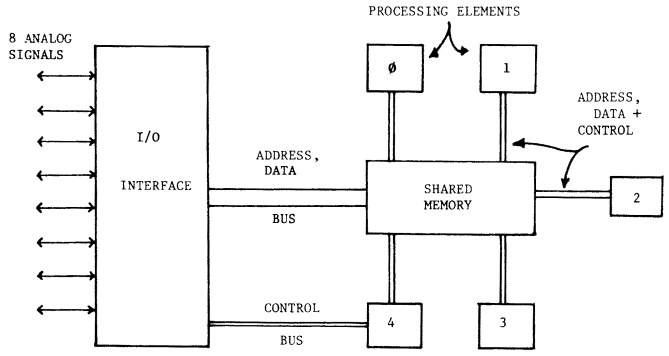

Fig. 6. Star configuration for the multiprocessor system.

TABLE II
MAIN CHARACTERISTICS OF THE MULTIPROCESSOR SYSTEM

	Processing Element Number				
	0	1	2	3	4
M_i (bytes)	3K	3K	3K	3K	32K
M_{si} (bytes)	1K	1K	1K	1K	1K
T_{ti} (usec)	39	39	39	39	5

M_i = Memory size for the ith PE
M_{si} = Shared memory size for the ith PE
T_{ti} = Minimum time to transfer one byte to shared memory
N_E = 5 (number of PE's)

star configuration (Fig. 6). The central PE maintains a partitioned data base that is shared by all processors. This arrangement supports direct interprocessor communication and allows for a simple control structure and high processing speed [22]. The architecture also allows for easy expansion and flexibility, as will be discussed subsequently.

D. System Description

The key feature of the system hardware is the mechanism by which the peripheral PE's request communication from the central processor to update the shared data base. The peripheral PE halts its own operation and remains in a halted mode until the central processor recognizes, services, and restarts it. This arrangement allows very simple control routines in the central processor, amounting to servicing the peripheral PE as merely an interrupting device, much as a printer or terminal interface.

Each peripheral is composed of RAM (1K bytes), EPROM (2K bytes), CPU, and interface devices (Table II). The RAM in each peripheral is mapped into the central processor's addressing space under control of the central processor. The central processor is allowed to address the peripheral processor's memory as if it were part of the central processor's memory map, thus allowing the data transfers to take place in parallel fashion in a minimum amount of time. Fig. 7 shows the block diagram for the interface of the central processor to one peripheral PE (labeled PERIPH X, where X stands for 0, 1, 2, or 3).

The central processor has complete control of the peripheral processors via the Reset, Halt, and buffer control lines. This allows the central processor to initialize the peripheral processors and to transfer data to and from the peripheral memories. The peripheral processor, however, must initiate the service by requesting it through the interrupt request signal. This technique allows faster communication and

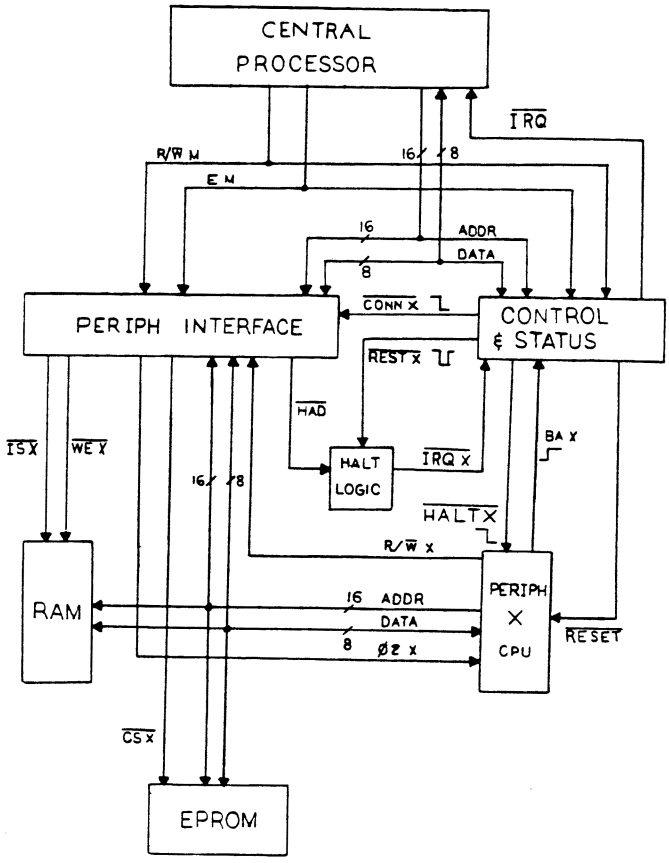

Fig. 7. Block diagram for the interface of the central processor to one peripheral processor.

simpler service software, since the peripheral processor has already given up control of its buses when the request is made. The interrupt driven service method also frees the central processor to perform the I/O functions for the system in the foreground, while servicing the peripheral processors in the background.

The processors chosen for use in this system are Motorola 6800 series processors, the central processor being a 6809 housed in an Exorciser development system, and the peripheral processor CPU's being General Motors Custom Microcomputer (GMCM) units. The GMCM processors are very similar to the 6809E microprocessor in that the clock source is external to the chip. This allows the peripherals to use the same clock as the central processor thus simplifying the interface design. The GMCM processors have an instruction set that is compatible with the Motorola 6801 processor; thus software compatibility is maintained. The peripheral PE's are constructed on prototype boards which are housed in the Exorciser unit along with the central processor and the shared data base memory.

E. Processor Interface Details

The address bus, data bus, and control lines of each peripheral processor are connected to the central buses through bus buffers which are controlled by the central processor through a parallel interface. When no data transfers are taking place, each processor has control of its own buses. When any peripheral requests service, the central processor connects the requesting peripheral's buses to its own and in effect moves the shared memory of the peripheral into its addressing space.

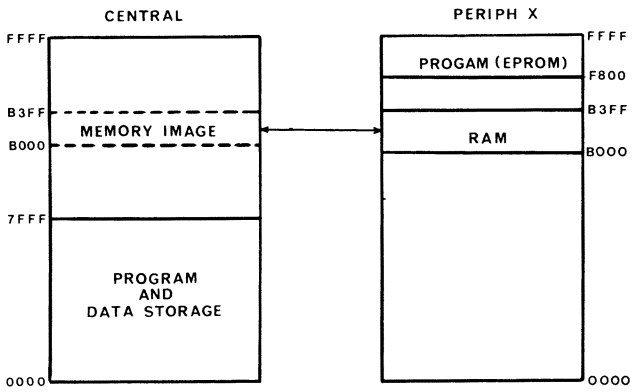

Fig. 8. Memory map for the central and peripheral processors.

The peripheral interface allows the central processor to read and modify the peripheral memory, transferring the needed data between the peripheral memory and the central memory. The interface only allows the central processor access to the RAM of the peripheral processor, since this is the only medium for data interchange which both processors are able to read and modify. The memory map for this system is shown in Fig. 8. The map shows the effect of the central processor connecting the peripheral processor buses to its own, producing a "peripheral memory image" in its address space. Note that, since only one peripheral may be serviced at one time, the peripheral memory images all fall at the same addresses. This allows the hardware for each peripheral processor to be identical, greatly increasing flexibility and expandability of the system.

E. Software Structure

The central processors serve the peripherals as if they are interrupting devices such as parallel interfaces. Thus, if more than one peripheral request service at one time, the central processor must arbitrate the requests. By software polling, the peripheral processsors are served by the central processor on a priority basis. The priority is determined by the order in which the peripherals are polled, with the highest priority peripheral being polled first, as illustrated by the flow chart in Fig. 9(a).

The flowchart for a typical peripheral program is shown in Fig. 9(b). The peripheral processor executes its dedicated calculation program, then halts itself and requests service from the central processor. After the service is completed, the central processor restarts the peripheral processor and the loop is repeated.

The development system requires that the central processor load the peripheral programs into the peripheral RAM as a part of the system initialization. This allows easy program modification and use of the resident assembler facilities of the central Exorciser system. In a dedicated system, the peripheral programs would reside in the EPROM provided.

A typical communication sequence between one peripheral processor and the central processor involves the following steps. (See Fig. 10 for timing relationships.)

1) A peripheral processor finishes its computation cycle and halts itself (at time T_1 on Fig. 10) sending its bus available line (BA) high. This halting action causes an interrupt request ($\overline{\text{IRQ}}$) generated by the peripheral interface to be sent to the central processor.

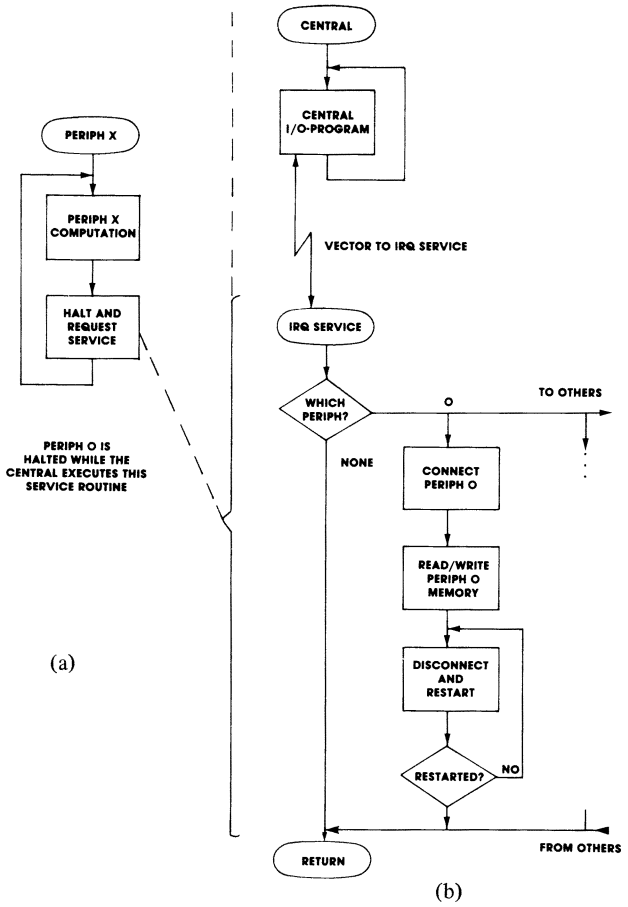

Fig. 9. (a) Central processor flow chart. (b) Peripheral flow chart.

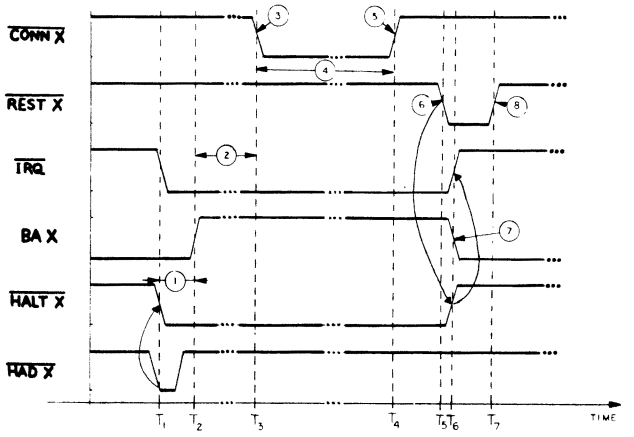

Fig. 10. Timing relationships for a typical communication sequence.

2) The central processor recognizes that a peripheral has requested service and starts executing an interrupt service route (T_2-T_3) which polls the peripherals to determine which one has requested service. If more than one peripheral has requested service simultaneously, the highest priority peripheral will be serviced first until all pending requests are serviced.

3) The central processor connects the requesting peripheral bus to the common system bus by sending the appropriate connect ($\overline{\text{CONNX}}$) line low (T_3).

4) The central processor then does the data transfers (READ and WRITE) which are appropriate for the particular peripheral (T_3-T_4).

5) After all data is transferred, the central processor releases the peripheral from the system bus by setting the appropriate connect line ($\overline{\text{CONNX}}$) high (T_4).

6) The central processor restarts the peripheral by setting the appropriate restart line ($\overline{\text{RESTX}}$) low, thus signaling the peripheral to restart a new computation cycle (T_5).

7) The peripheral processor restarts from its halted condition, taking control of its own buses and sending its bus available (BA) line low (T_6).

8) The central processor recognizes that the peripheral has restarted by monitoring the bus available line. When the restart has occurred, the central processor resets the restart line high (T_7) to be ready for the next service request.

At this point, the central processor returns to its main I/O program. If another peripheral has requested service while the first was being serviced, the central processor is immediately interrupted and the whole process starts at Step 2 again.

The service routines allow any number of bytes of data to be transferred to or from any peripheral. This, along with some mapping of the central shared data base, allows any piece of data to be transferred from any peripheral or the central processor to any other peripheral.

G. Deadlock Avoidance

Deadlock occurs when one processor has control of the bus system and is waiting to communicate with another processor which is waiting to access the bus and cannot be accessed. This condition cannot occur on the system presented because the central processor has control of the system bus and arbitrates all simultaneous requests for access. A form of resource hogging can occur, however, if two processors finish their computations and request service in nearly the time it takes to service one processor. This results in the central processor always servicing these two peripherals to the exclusion of all others. If only one peripheral finishes in a time comparable to the service time, that processor must be the lowest priority to insure that all others get serviced without inordinate waits.

V. IMPLEMENTATION OF SIMULATION

The model is already partitioned into four functional blocks. Each block was assigned to one PE as defined in the previous section. An additional PE was used to handle the inputs to the system and output from the system.

A. Interprocessor Communications

This refers to the instances of time in which the PE's interchange messages. The processing elements perform all message transfers at the *same rate*. Fig. 11 shows the timing diagram that corresponds to this message transfer technique. The advantage of this method is that the communications protocol is easy to implement. The disadvantage is that some PE's would have to wait a certain length of time before the message transfer is done unless all blocks have the same processing time. For real-time simulation, the following relationship has to be satisfied:

$$\text{Max}\,(T_i + T_{ti}) + \leqslant T_s \qquad i = 1, 2, \cdots, n \tag{16}$$

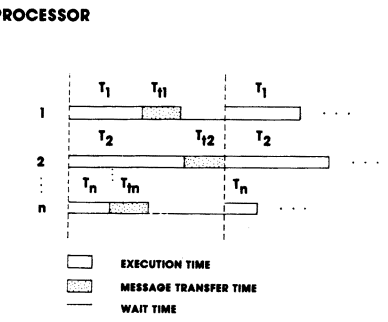

Fig. 11. Timing diagram for a system with single sample rate message transfer.

TABLE III
NORMALIZATION FACTORS FOR THE VARIABLES USED IN THE SIMULATION

VARIABLE (x)	NF_x	
Throttle angle (α)	2.83	$(\text{degrees})^{-1}$
Air flow rate (AR)	56.986×10^3	$(\text{lb/sec})^{-1}$
Fuel flow rate (FR)	56.986×10^3	$(\text{lb/sec})^{-1}$
EGR flow Rate (ER)	56.986×10^3	$(\text{lb/sec})^{-1}$
Manifold pressure (PM)	16	$(\text{psi})^{-1}$
Air mass (MA)	56.986×10^3	$(\text{lb})^{-1}$
Indicated torque (TI)	32	$(\text{lb-ft})^{-1}$
Moment of inertia (J_T)	8192	$(\frac{\text{lb-ft-sec}}{\text{RPM}})^{-1}$
Engine speed (N)	1	$(\text{RPM})^{-1}$

where T_{ti} is the duration of the message transfers for the ith PE, T_s is the sampling period, and T_i is the processing time for the ith PE.

A single sample rate method for interprocessor communication was used because it simplifies the discretization process for the model. Synchronization of time steps was achieved by introducing delays on each PE so that the sum of their computation time plus transfer time was the same for all PE's. The sampling interval was chosen to be 4/1024 s which is small enough to provide good results even at high engine speeds. The programming was done using assembly language to optimize execution speed. All simulation calculations were done using integer arithmetic. Some internal calculations were done using 24-bit precision to increase accuracy (manifold pressure, for example).

A major problem that arises when writing an assembly language program with integer arithmetic on a microcomputer is the normalization of variables. The microcomputers used have 8-bit and 16-bit precision, and all the variables had to be scaled to fit in either precision. This was accomplished defining a normalization factor N_F defined as follows:

$$NF_x = \frac{x_{cu}}{x_{eu}} \qquad x_{cu} \tag{17}$$

where x_{cu} is the variable expressed in computer units and x_{eu} is the actual variable expressed in engineering units. Table III shows the normalization factors used for the simulation.

Some engine characteristics (such as throttle) and the influence of the interaction of several variables are readily

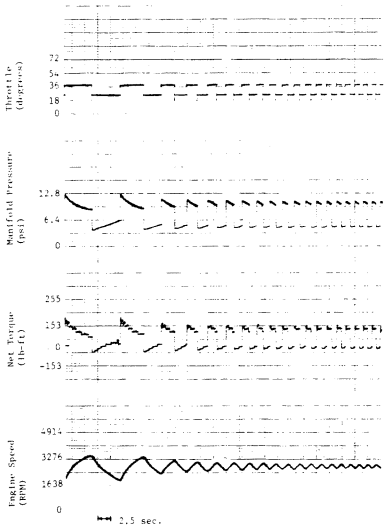

Fig. 12. Typical simulation output.

TABLE IV
MEMORY SIZE AND EXECUTION TIMES FOR THE DIFFERENT PE'S

	Processing Element Number				
	0	1	2	3	4
	Carburetor	Intake Manifold	Combustion	Dynamics	I/O
M_{pi} (bytes)	272	599	435	431	1415
T_{pi} (msec)	.22	3.2	2.35	3.8	----
T_{ai} (usec)	270	480	320	290	----

M_{pi} = Program memory size for the ith PE
T_{pi} = Program execution time for the ith PE
T_{ai} = Actual message transfer time for the ith PE

available in tabulated form obtained from engine dynamometer data. For example, the influence of spark advance and A/F on indicated torque in Fig. 4 was obtained from engine dynamometer tests and stored in memory for several operating conditions. The influences of these variables were implemented using subroutines based on lookup tables.

As shown in Fig. 4, the model involves the use of functions of two variables, for example, the effect of speed and charge density on torque. The implementation of such functions was done using lookup tables having two indexes (one for each variable). The first index is used to point to one of several tables and the second index is used to point to a specific entry in the table chosen.

B. Simulation Results

A typical output trace from the engine simulation is shown in Fig. 12. The graphical output shows the real-time response of the simulation to a square wave throttle input. Note the dynamic response characteristics of the engine speed and manifold pressure. Within the limits of the complexity of the mathematical model used, these traces correspond to the results shown by Dobner for the mainframe executed version of the model. Dobner's results, in turn, correlate with actual engine data.

The model mathematics require that the slowest functional unit take no more than 4 ms to execute its program. Table IV shows the actual execution times for the funtional units, as well as the required communication times. The system does achieve real-time performance, and with further optimization, could perform even better. This would allow the simulation sampling interval to be shortened, thus increasing accuracy.

TABLE V
SPEED COMPARISONS WITH OTHER SYSTEMS

Configuration	Speed
Authors multiple 6802/09 (1 MHz)	1
Single 68000 (10 MHz)	1.38
Single Z8002 (6 MHz)	0.79
Multiple 68000 in same architecture (10 MHz)	5
VAX - 11/780 (5 MHz)	1.38
PDP - 11/70 (7.5 MHz)	1.11

Higher speed values mean faster systems.

VI. COMPARISON WITH OTHER PROCESSORS

Table V shows a comparison in terms of speed of the multiple microcomputer system against other state-of-the-art high-performance microprocessors and minicomputers. Execution times figures used to obtain table estimates were obtained from a comparison made by Patterson and Sequin [23]. A 10-percent overhead for message transfer times was assumed, which is in accordance with Table IV.

From Table V, one can see that the results presented in this paper could have been obtained by using a single high-performance microcomputer or minicomputer. However, the system presented here provides a very cost-effective solution to the real-time simulation problem.

The main advantage of multiple microcomputers is that, as technology advances, one can incorporate state-of-the-art processors in the same architecture, thus achieving higher throughput.

VII. ECONOMIC JUSTIFICATION

As pointed out previously, the results presented in this paper could have been obtained using high-performance minicomputers such as a PDP/11 or a VAX/780. However, the cost of these systems was prohibitive for the particular application that is described here. Consequently, cost was a major factor for the design of the real-time simulator hardware.

The multiple microcomputer system presented in this paper is a low-cost system. The cost values presented in this section will be for a complete processing element except power supplies and enclosures. The hardware devices (with their associated costs involved in a PE) are shown below (in U.S. dollars):

Board	62.00
Processor	10.95
Bus Transceivers	6.30
Memory	9.80
Additional Circuitry	3.05
Total	92.10

The multiple microcomputer system could be packaged in a single self-contained enclosure as a general purpose real-time simulator. All programming could be done off-line in another development system, and the programs could be

down-loaded to the simulator. The main advantage of this approach is that by changing some Programmable Read Only Memories (PROM's) different engines (or different systems) could be simulated. The cost of such a compact simulator is estimated to be approximately $1000, which is far less than the cost of a typical general purpose computer that could be used to do the same job.

VIII. MULTIPROCESSOR PERFORMANCE

When working with multiprocessor systems, it is sometimes useful to compare them with uniprocessor systems in terms of speed. A performance measure has been developed [24] for the system discussed in this paper which is based on a) execution times to evaluate a certain family of algorithms, and b) message transfer times between processors in terms of bytes transferred. The family of algorithms referred to previously is of the inner product form which is used extensively in simulation. A generic term is of the form

$$y(k) = a_1 x_1(k) + a_2 x_2(k) + \cdots + a_M x_M(k). \tag{18}$$

The main result of the performance study is that the multiple microcomputer system performs approximately N times faster than an equivalent uniprocessor system provided the number of terms M is large (about 64 or more), where N is the number of PE's in the configuration. If N is constant and M increases, the performance increases. Conversely if M is constant and N increases, the relative performance decreases due to the overhead involved in the message transfer times.

IX. OTHER APPLICATIONS

The hardware design used to implement the engine model lends itself to any multiprocessing task which requires few variables to be passed between functional units. The design also allows more peripheral processors to be added easily, only requiring that control lines be attached. All peripherals are identical in hardware detail, making possible integration of the peripheral processors on a single chip or small board. Each peripheral can be treated as a dedicated co-processor, allowing functional units to be added or removed as the application warrants.

For example, an engine controller could be made of a central processor that handles I/O functions and a number of peripherals that each handle one function such as fuel control, spark control, driveline control, etc. This type of arrangement would allow minimum systems to be built from the same building blocks as full-feature controllers. This would save development time and allow the user of a controller to configure the system as necessary from the appropriate blocks.

X. SUMMARY

A multiprocessor system was presented which was designed to run a simulation of an internal combustion engine in real time. The system is composed of a central processor and four peripheral processors in a star configuration. The hardware design, software structure, and application constraints were discussed. Results of the engine simulation application were presented. Further application possibilities were suggested, including flexible modular engine control systems.

ACKNOWLEDGMENT

The authors would like to thank Dr. J. Olin for his cooperation in the procurement of hardware and facilities for this research, and for his continued support. Also, the support of General Motors Institute, Delco Electronics, and General Motors Research Labs is greatly appreciated. This project was the effort of other former GMI students as well, whose participation is acknowledged. The aid of Dr. D. Dobner was invaluable in applying his model to the hardware system. Finally, the comments and suggestions of the reviewer were very helpful.

REFERENCES

[1] A. B. Bortz, "Computer simulation of engine control systems—A tool for identifying critical components and interactions," presented at the Soc. Automotive Engineers Int. Congr. Exp., Detroit, MI, Feb. 1980, paper no. 800055.

[2] B. Mitchell, R. Abrams, and R. A. Scott, "All-digital simulation of simple automobile maneuvers," *Simulation*, vol. 37, pp. 179–186, Dec. 1981.

[3] D. B. Gilmore, "Analog simulation of a hybrid gasoline-electric vehicle," *Simulation*, vol. 38, pp. 85–92, Mar. 1982.

[4] W. B. Gallis, "A microprocessor system simulation of engine control modules," presented at the Soc. Automotive Engineers Int. Congr. Exp., Detroit, MI, Feb. 1981, paper no. 810450.

[5] L. D. Reid and W. O. Graf, "The fitting of linear models to driver response records," presented at the Soc. Automotive Engineers Int. Congr. Exp., Detroit, MI, Feb. 1982, paper no. 820304.

[6] A. M. Kotwicki, "Dynamic models for torque converter equipped vehicles,' General Motors, Warren, MI, Res. Pub. GMR-3886, Nov. 1981.

[7] J. A. Tennant, R. A. Giacomazzi, J. D. Powell, and H. S. Rao, "Development and validation of engine models via automated dynamometer tests," presented at the Soc. Automotive Engineers Int. Congr. Exp., Detroit, MI, Feb. 1979, paper no. 790178.

[8] R. L. Morris, H. G. Hopkings, and R. H. Borcherts, "An identification approach to throttle-torque modeling," presented at the Soc. Automotive Engineers Int. Congr. Exp., Feb. 1981, paper no. 810448.

[9] J. F. Cassidy, M. Athans, and W. H. Lee, "On the design of electronic automotive engine controls using linear quadratic control theory," *IEEE Trans. Automat. Contr.*, vol. 25, pp. 901–912. Oct. 1980.

[10] D. J. Dobner, "Dynamic engine models for control development—Part I: Nonlinear and linear model formulation," General Motors, Warren, MI, Res. Pub. GMR-3783, Jan. 1982.

[11] F. E. Coats, Jr. and R. D. Fruechte, "Dynamic engine models for control development–Part II: Application to idle speed control," General Motors, Warren, MI, Res. Pub. GMR-3789, Jan. 1982.

[12] C. Weitzman, *Distributed Micro/Minicomputer Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1980.

[13] S. H. Fuller, J. K. Ousterhout, L. Raskin, P. I. Rubinfeld, P. J. Sindhu, and R. J. Swan, "Multi-microprocessors," *Proc. IEEE*, vol. 66, pp. 216–228, Feb. 1978.

[14] G. A. Anderson and E. D. Jensen, "Computer interconnection structures: Taxonomy, characteristics and examples," *ACM Computing Surveys*, vol. 7, pp. 197–213, Dec. 1975.

[15] W. C. Mcdonald and R. W. Smith, "A flexible distributed testbed for real-time applications," *Computer*, pp. 25–39, Oct. 1982.

[16] P. R. Ma, E. Y. S. Lee, and M. Tsuchiya, "A task allocation model for distributed computing systems," *IEEE Trans. Comput.*, vol. C-31, Jan. 1982.

[17] B. P. Buckles and D. M. Hardin, "Partitioning and allocation of logical resources in a distributed computing environment," General Res. Corp., Huntsville, AL, 1979.

[18] H. S. Stone, "Multiprocessor scheduling with the aid of network flow algorithms," *IEEE Trans. Software Eng.*, vol. SE-3, pp. 85–94, Jan. 1977.

[19] V. B. Gylys, and J. A. Edwards, "Optimal partitioning of work-

loads for distributed systems," in *COMPCON Dig. Pap.*, Fall 1976 IEEE cat. no. EHO 15H.
[20] D. J. Dobner, "A mathematical model for development of dynamic engine control," Society of Automotive Engineers, paper 800054, Feb. 1980.
[21] P. H. Enslow, Jr., "Multiprocessor Organization—A Survey," *ACM Computing Surveys*, vol. 9, no. 1, Mar. 1977.
[22] J. R. Pimentel and M. Loeffler, "An exorciser-based multimicrocomputer system for distributed processing applications," General Motors Inst., Elec. Eng. Res. Rep. no. 2, Jan. 1982.
[23] D. A. Patterson and C. H. Sequin, "A VLSI RISC," *Computer*, pp. 8–18, Sept. 1982.
[24] J. R. Pimentel, "Performance of a multiple microcomputer system for distributed processing in control applications," presented at the 25th Midwest Symp. Circuits Systems, Michigan Tech. Univ., Aug. 1982.

A Microcomputer-Controlled Powertrain for a Hybrid Vehicle

CLEMENT B. SOMUAH, MEMBER, IEEE, ANDREW F. BURKE, BIMAL K. BOSE, SENIOR MEMBER, IEEE, ROBERT D. KING, MEMBER, IEEE, AND MICHAEL A. POCOBELLO

Abstract—**The design and testing of a microcomputer-controlled powertrain for a hybrid (heat engine/electric) vehicle are described in this paper. The detailed control strategy used and its software implementation on the Intel 8086 microcomputer are discussed. The powertrain is made up of an electric motor, gasoline engine, automatic transmission with the torque converter removed, engine and electric motor clutches, and Hy–Vo transfer chains. The drive system has been installed in vehicles and tested on a chassis dynamometer. Test results show smooth shifting of the transmission and sequencing of the clutches in the different hybrid operating modes. The overall performance of the hybrid car is good, and driveability is comparable to a conventional automobile.**

I. INTRODUCTION

HYBRID passenger cars, which utilize both an electric motor and a gasoline engine, could become an attractive alternative to the conventional ICE-powered automobiles if gasolines prices continue to rise and/or the gasoline supply becomes uncertain. The use of all-electric vehicles would, of course, eliminate completely the need for gasoline/diesel fuels for powering on-road vehicles. However, electric vehicles have performance and range limitations. Including a small gasoline engine in the driveline, as in the vehicle in this paper, offers a means of overcoming these limitations.

The objectives of the United States Department of Energy's Near-Term Hybrid Vehicle Program was to design, build, and test a full-size hybrid car that would provide the same utility and comfort as a conventional full-size car but, in addition, minimize the use of petroleum and utilize, instead, wall-plug electricity. To meet this goal, a microcomputer-controlled hybrid powertrain was designed and built [1]. The use of a microcomputer was necessitated by the degree of complexity of the overall control system for the powertrain.

Manuscript received August 30, 1982; revised December 8, 1982. This work was supported by the U.S. Department of Energy, Office of Electric and Hybrid Vehicles, and was managed for the D.O.E. by the Jet Propulsion Laboratory.

C. B. Somuah is with the University of Petroleum and Minerals, Dhahran, Saudi Arabia.
A. F. Burke, B. K. Bose, and R. D. King are with the General Electric Company, Research and Development Center, Schenectady, NY.
M. A. Pocobello is with Triad Services, Troy, MI.

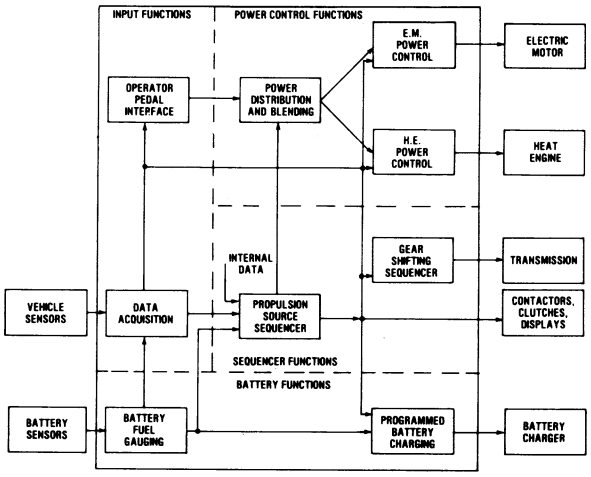

Fig. 1. Hybrid vehicle controller functions.

The major vehicle functions under microcomputer control are shown in Fig. 1. Propulsion sequencing is concerned with all the decisions associated with transitions from one mode of operation to another, such as closing/opening of electrical contactors and clutches and turning on/off the heat engine and electric motor. Transmission shifting is also under the control of the microcomputer. The optimum gear of operation is dependent on the vehicle speed, propulsion units operating, and the power demand. The electric motor clutch is modulated by the microcomputer using closed-loop feedback control for smooth vehicle operation.

Section II of the paper gives a brief description of the powertrain and a discussion of its different modes of operation. The transmission shifting strategy and sequencing algorithms are discussed in Sections III and IV of the paper. The control strategy and algorithms for controlling the modulation of the vehicle drive clutch are described in Section V.

0278-0046/83/0500-0126$01.00