# SOFTWARE IN DEVELOPMENT FLIGHT TEST PACKAGES

**Leon W. Maschoff**
**Staff Member**
**TM Advanced Development Division**
**Sandia National Laboratories**
**Albuquerque, New Mexico**

## ABSTRACT

This report discusses application software options that are available for the design of development flight test microprocessor packages. Prelaunch parameter updates, telemetry channel calibration, system and processor self-test, raw data store, reduced data store, and telemetry formatting are some of the on-board functions that software can perform in addition to the customary control and arithmetic calculation tasks. A specific application that implements these functions as software modules and the hardware needed for support is described. The hardware in this application includes a 16-bit microprocessor that is dedicated for real time operations and an 8-bit microprocessor that services the self-test functions.

## INTRODUCTION

Software can serve many functions in the design and testing of system development instrumentation packages. This report identifies some of these functions and describes their use for a specific application.

The intent of development testing is to learn as much as is practical about a system's behavior when that system is subjected to test conditions and environments that it eventually will encounter in use. Results may identify follow-on test parameter requirements or may prove the test concept totally invalid as proposed. The objective of flight test packages is to measure environmental and performance data that will characterize the system performance if the system functions as predicted or if it fails. Data channel allocation might best be determined by predicting failure modes.

The process for determining design features and the parameters to be measured is an iterative exchange between the experimenter and the instrumentation designer. Final agreement may not be reached until late in the development schedule. That is why a built-in flexibility for change is desirable.

The design of a test package requires early consideration of the hardware and software functions and interfaces. Chip count, response time, power requirements and flexibility are important considerations. Once hardware is defined and support drafting and layouts are completed, it becomes more difficult to make significant changes; however, software can be changed readily if the original design anticipates sufficient memory and Input/Output (I/O) interfaces. Thirty-two kilobit Erasable Programmable Read Only Memories (EPROMs) are readily available and denser equivalents are being developed. Random Access Memory (RAM) for both reading and writing are also available in capacity and speeds that satisfy most needs. Most available RAMs are volatile i.e., they retain data only while power is present. The available non-volatile RAMs are not competitive to volatile RAMs in read/write speeds. Given a processor choice, I/O needs, and a crude estimate of program size, the hardware design and layout can be completed. Software can be sketchy other than identifying modules and functions. System changes might then be limited to software updates.

Software functions for test packages can be classified as real time and nonreal time. Real time is used here to suggest that software which executes using data just acquired by sampling the output from some onboard sensor. Real time execution occurs on a sampled basis and processes current data. The same code is executed repetitively with cycle times fixed (synchronous) or dependent on the number of functions performed in one sample interval (asynchronous). The approach used to implement real time software depends on how critical it is to have optimal cycle times. Software for the slow speed version can be written in a higher level, less efficient language than assembly language. When speed is required, software must be written in the most efficient language and hardware must be used to enhance that speed objective. Real time processors are usually used as some part of a control system. The processor may make decisions on raw data or it may make decisions based on mathematical computations involving that raw data.

Nonreal time software is usually executed one time on request and has a defined starting location and stopping point. Execution time is not critical nor is the efficiency with which it is coded. Most nonreal time flight package software serves as a tool for testing, interaction, or data output.

Some specific software categories for flight test applications are: telemetry formatting, data scaling, data conversion, data sampling, calibration, self-test, data reduction and pre-test updates. These categories are in addition to the software that directly supports the test algorithm. Telemetry formatting includes setting channel assignments and data update rates. As long as hardware compatibility is maintained, software can scale the data with a simple shift, multiply or divide. System calibration, self-test and pre-test updates are essential for flight tests where the tests are not easily repeated because they can assure data integrity and establish confidence that the system is reliable. Self-test exercises the

complete data system from input sensor to output control. If sensor data can be simulated over the vehicle's predicted flight profile, the test is even more informative. Prelaunch updates require a data terminal communication link to the package via the vehicle umbilical cable. Data reduction and conversion to engineering units are software tools that are essential for early laboratory testing and post-test evaluation because it is difficult to evaluate real time data when that data is expressed in a hexadecimal, octal, or binary format.

## AN APPLICATION

### Test Objective

A flight test application that will be used for describing software needs and alternatives is one whose overall objective is to "guide a surface launched projectile during the terminal phase of its trajectory". The nominal ballistic parameters are:

1. launch velocity of 1400 feet/second
2. altitude at apogee of 16000 feet
3. time of flight of 40 seconds
4. ground range of 20000 feet
5. roll rate from 4 Hz to 12 Hz.

### Requirements

An optical photodiode array mounted in the vehicle's nose scans the target area for a light source. Illumination of one or more of the array elements provides angle data from the vehicle spin axis to the target. An on-board magnetometer provides a measure of vehicle roll position from the launcher reference. Trajectory alteration is provided by deflecting an inertially stabilized ring canard which is attached to a roll controlled nose section. The nose section rolls on a cylindrical center shaft which is connected to the afterbody. The optical array is positioned inside the forward end of this shaft. DC motors control the nose roll position and canard deflection. An on-board microprocessor processes the array data, measures current roll position of the afterbody by examining magnetometer output, determines relative roll position between the nose and afterbody, determines the deflection of the canard, solves the six-degree-of-freedom trajectory equations for the projectile, and generates appropriate guidance commands to deflect the canard and position the nose to guide the projectile towards the target. Some combination of hardware and software is required to translate the input signals to reasonable output control signals and while doing so generate data for telemetry and on-board storage. A real time software package is needed that operates at rates greater than 200 Hz or less than 5 millisecond cycle time. Required numerical computation accuracy dictates a need for at least a 16-bit processing

capability. The processor chosen to support this application is the Motorola MC68000. One of its desirable features is that it has 32-bit data and address registers.

### Hardware

A block diagram of the processor package and its interfaces are shown in Figure 1. The magnetometer, eyeball, deflection pot and nose roll position are system inputs while the nose and deflection drive are the control system outputs. Data outputs for telemetry are identified.

The processor package memory board supports the six 4K by 8-bit ERPOMs that are used for program and fixed data storage, the four 2K by 8-bit RAMs and the multiplier. All are capable of being accessed at the 8 MHz 68000 rate without having to insert fixed delays. The multiplier and the analog processor are examples of hardware enhancements to software for improving processing speed.

The 8085 processor serves the self-test function by generating data to simulate vehicle performance.

## SOFTWARE IMPLEMENTATION

Given the required inputs, outputs, hardware, test conditions and the algorithm, there are some building blocks to consider as well as some choices to make.

### Arithmetic

A Fortran or equivalent high-level language program makes internal variable representation transparent to the user. System software takes care of positioning the decimal point and all the related bookkeeping chores. When system software does not exist or when it does and it cannot be used because of time or memory constraints, an alternative approach is required. Without system software other than the built-in instruction set, the MC68000 system provides a 16-bit data bus, a 23-bit address bus, instructions for 16-bit multiply and divide, 32-bit add and subtract, and the usual compliment of support instructions. Internal data and address registers are 32-bit. A single instruction can access an 8-bit byte, a 16-bit word, or a 32-bit long-word. With the data bus only 16 bits wide, a 32-bit request exercises the data bus twice. Thirty-two bits of data represents an integer magnitude of $2 \times 10^9$. The data magnitudes encountered by the algorithm range between small fractional values to large integer values. A fixed-decimal point arithmetic is used for this application where XXXX.XXXX represents a 32-bit hexadecimal number with decimal point positioned as shown. This fixed point representation enables fast computation and because

the decimal point is on the 68000 word to long-word boundary some bookkeeping is saved.

**Special Functions**

Given a choice of arithmetic, the next consideration is the evaluation of functions that are not a part of the instruction set. An overall estimate of the algorithm operations yields;

| | |
|---|---|
| 175 | add/subtract |
| 15 | divide |
| 250 | multiply |
| 7 | sin x |
| 7 | cos x |
| 2 | arc sin y |

Adds and subtracts are part of the instruction set. Their average speed at 8 MHz is about 1.2 microseconds. Multiplies and divides are also part of the instruction set but they are for 16-bit arguments. There are some products requiring multiplies with both integer and fraction. A predicted multiply time of 8 microseconds for unsigned 16-bit variables indicates that sample speed limitations may result from the required number of multiplies. Thirty-two by 32-bit products evaluated with software takes 60 microseconds. An average multiply time of less than 15 microseconds is needed to achieve the 5 millisecond cycle time. Hardware multiply and pre-scaling or mode selection are methods used for gaining some advantage.

The hardware multiplier generates a product of two 16-bit numbers in 250 nanoseconds but the time needed to deliver the numbers to the multiplier and retrieve the final product makes a total multiply time of 7 microseconds. This compares with 8 microseconds for software. The saving is significant but not dramatic enough to solve the problem. Mode selection is pre-determining the size of the multipliers so that the fastest method is used. Let A.B and X.Y represent two 32 bit multipliers with decimal point in the middle. Then

$$\begin{array}{r} A.B \\ X.Y \\ \hline AX + (AY+BX) \times 16^{-1} + BY \times 16^{-2} \end{array}$$

is the product. If any of the four halves of the multipliers are not present, the product can be obtained quicker because fewer operations are required. Pre-scaling forces both multipliers to occupy 16 bits. The shifting before and after multiplying does introduce some error in the final product.

The standard method for evaluating sin x, cos x, and arc sin y requires evaluating terms in a power series expansion. Table look-up has been chosen for this application because enough memory is available for the tables and also because the look-up with linear interpolation is fast. The algorithm usually requires the sine and cosine of the same variable at the same time. The method used is as follows:

Divide the interval 0 to PI into x values where the x increment is $(0\text{-}0156)_{10}$ or $(400)_{16}$. This requires 202 table samples. Force all the entries to be positive.

The table entries are developed by:

| Relative Address | | Value |
|---|---|---|
| Long | 0, 1 | sin 0 |
| Word | 2, 3 | cos 0 |
| | 4, 5 | sin $(400)_{16}$ = sin $(.0156)_{10}$ |
| | 6, 7 | cos $(400)_{16}$ = cos $(0.0156)_{10}$ |
| | 8, 9 | sin $(800)_{16}$ |
| | 10, 11 | cos $(800)_{16}$ |

Thirty-two bits or one long-word contains the sin x and cos x value. /sin x/ and /cos x/ are always less than one. At x = 0, x = PI/2, and X = PI the values are forced to be $(FFFF)_{16}$. By choosing sample intervals spaced $(400)_{16}$ apart it is easy to address the interpolating interval.

If x = $(.64406)_{10}$ and x = $(A4E1)_{16}$ mask the $(A4E1)_{16}$ with $(FFFFFCOO)_{16}$ to give $(A400)_{16}$. Then (A4E1) - (A400) = $(E1)_{16}$. Shift the $(A400)_{16}$ right eight places to get $(A4)_{16}$. Then the contents of the address $(A4)_{16}$ plus the address of the look-up table is the sine of the lower bound while the contents of $(A8)_{16}$ plus the address is the upper bound. Linear interpolation gives sin x = .60043. The divide by $(400)_{16}$ is a simple shift so there are a minimum of arithmetic operations. The cos x computation is identical except it interpolates on the second half of the long-word. Sin x and cos x are both computed in less than 100 microseconds with this technique. Arc sin y is computed using the same basic approach. It computes in less than 50 microseconds.

**Functional Modules**

Application software can usually be defined as functional modules where a module performs a single or similar operation. A module characteristic is that it has a defined input and output so that it can be written and tested separately. The given application is divided into 11 separate software modules. They are:

Monitor
Data Recovery
Self-Test
Arm
Initialization
Magnetometer
Array Processing
Guide/Control
Integrate
Data Output

} real time algorithm

Figure 2 identifies the interaction between these modules and how they are accessed.

The monitor module permits communication between a data terminal and the processor package and includes the ability to examine and change contents of registers and memory (RAM). Single stepping, setting breakpoints, and tracing are also available. It resides in EPROM beginning at address $(0000)_{16}$ and uses 3K of memory space. Entry into monitor is accomplished through system power-up and system reset.

The data recovery module is a nonreal time function. Its prime purpose is for use as a testing tool and for post-test data recovery. It outputs data to the terminal in scaled decimal units.

Calibrate establishes the validity of the processor-to-telemetry interface by stepping through all of the D/A interfaces with predefined amplitude inputs. It also checks the integrity of the RAM and the address and data bus. Calibrate is accessed via an external interrupt from the control box or via a request from the monitor program.

Self-test tests the complete process and control system except for the magnetometer and eyeball lens array combination. Self-test is accessed via an interrupt initiated at the control box. Figure 3 shows the control box interface and the self-test controls. An Intel 8085 processor is used as the self-test controller. It responds to the self-test interrupt-pull-out combination by generating a varying amplitude and frequency sinusoid that simulates magnetometer behavior throughout the trajectory. At some point after apogee in this simulated trajectory, eyeball data is inserted so that the motor drives and TM channels are exercised with known input data. The communications bus between the 8085 and 68000 exchange start and stop commands as well as variations in self-test modes.

Arm enables both processors for launch. All of the self-test features are hardware disabled so that default to the test magnetometer and eyeball is assured. At arm, the 68000 enters the real time loop and begins processing the input data. At pull-out, time is reinitialized to

zero and the magnetometer data is ignored for a short time so the effects of motion on the launcher can be ignored until separation is assured.

The initialization module is really part of the real time algorithm. it is executed one time and sets up the RAM arrays and variables. Once completed, real time processing begins.

The magnetometer, array processor, guide/control, and integrate modules are all specific parts of the processing algorithm. The magnetometer module includes a 25 Hz low pass digital filter. The integrate modules include a fourth-order Runga-Kutta integrator for the six-degree-of-freedom equations. These two functions account for a major portion of the arithmetic computation. The chart in Figure 2 identifies the real time module relationships. Data output is part of the real time algorithm in that it too is executed each real time cycle. It determines which of the data items are to be output during the current cycle.

Modules are tested as stand alone routines until they are logically and computationally correct. Real time module testing is aided by using the A/D and D/A channels (Figure 1) for observing the dynamic behavior of the real time variables.

## SUMMARY

Software provides a desired flexibility to the design and building of instrumentation packages. The advantages of that flexibility are realized when changes can be made to the operational system without modifying hardware. Memory speed and density makes replacing hardware with software even more attractive.

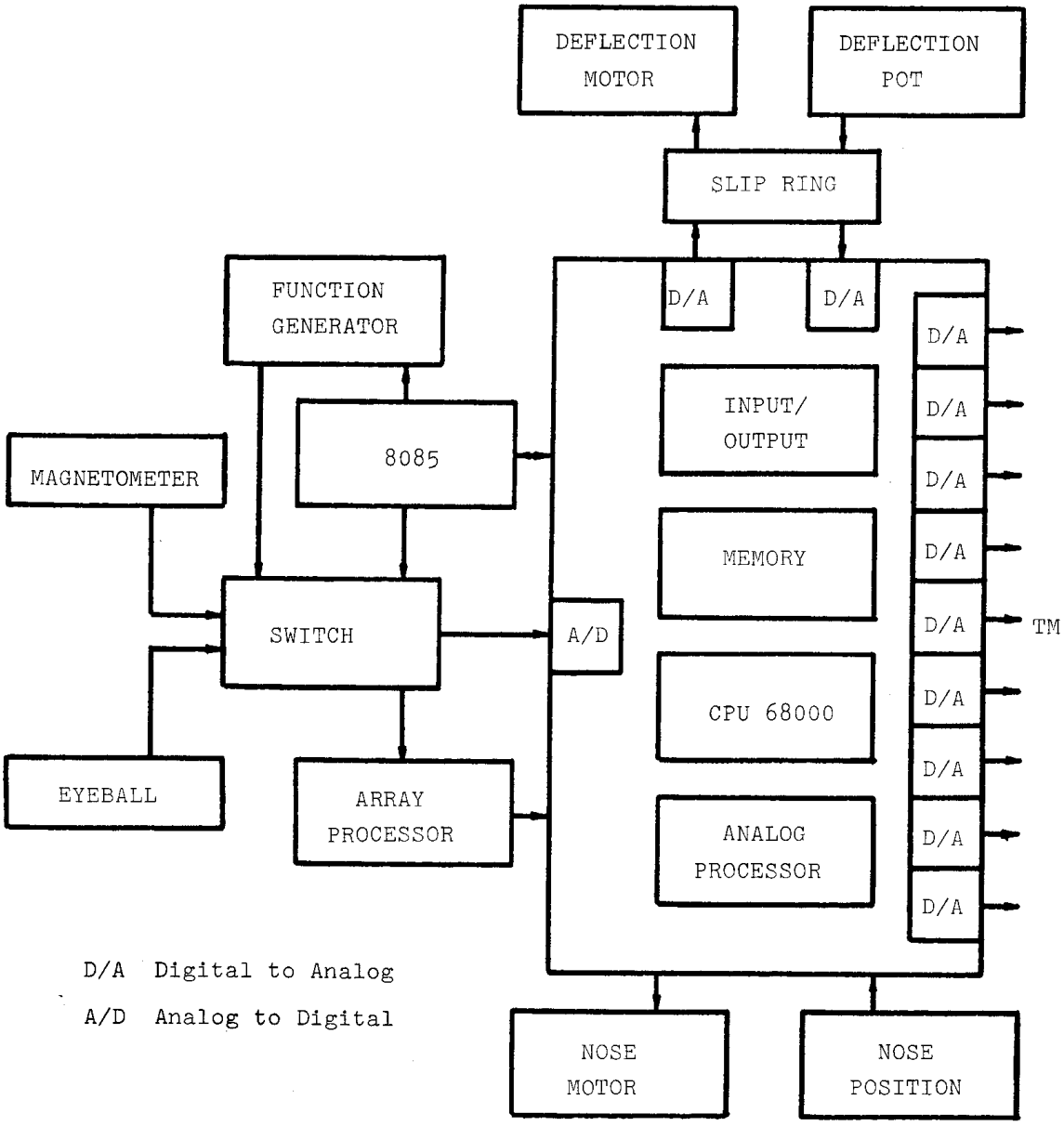


**Figure 1**

**Processor Package and Interfaces**

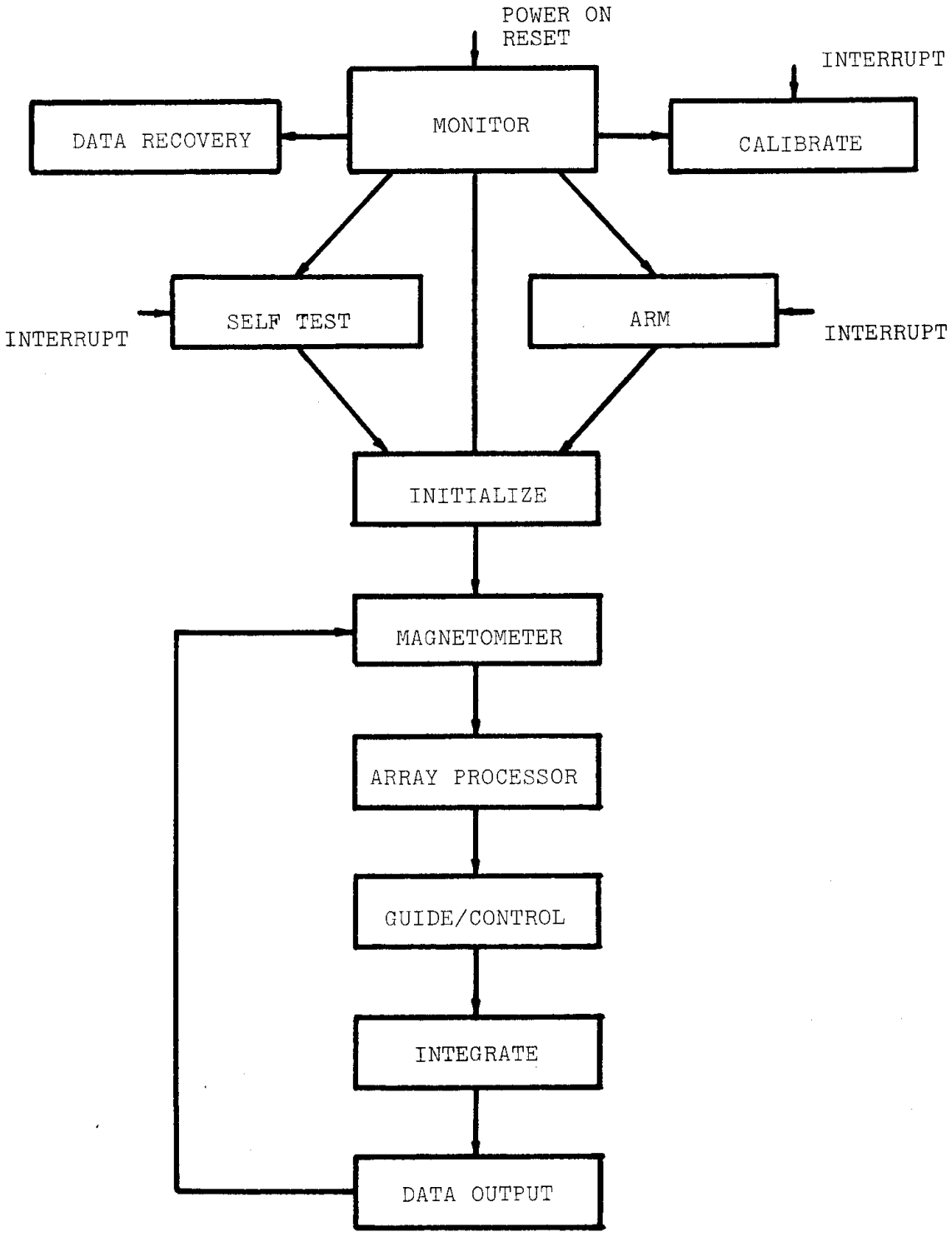


**Figure 2**

**Software Module Interaction**

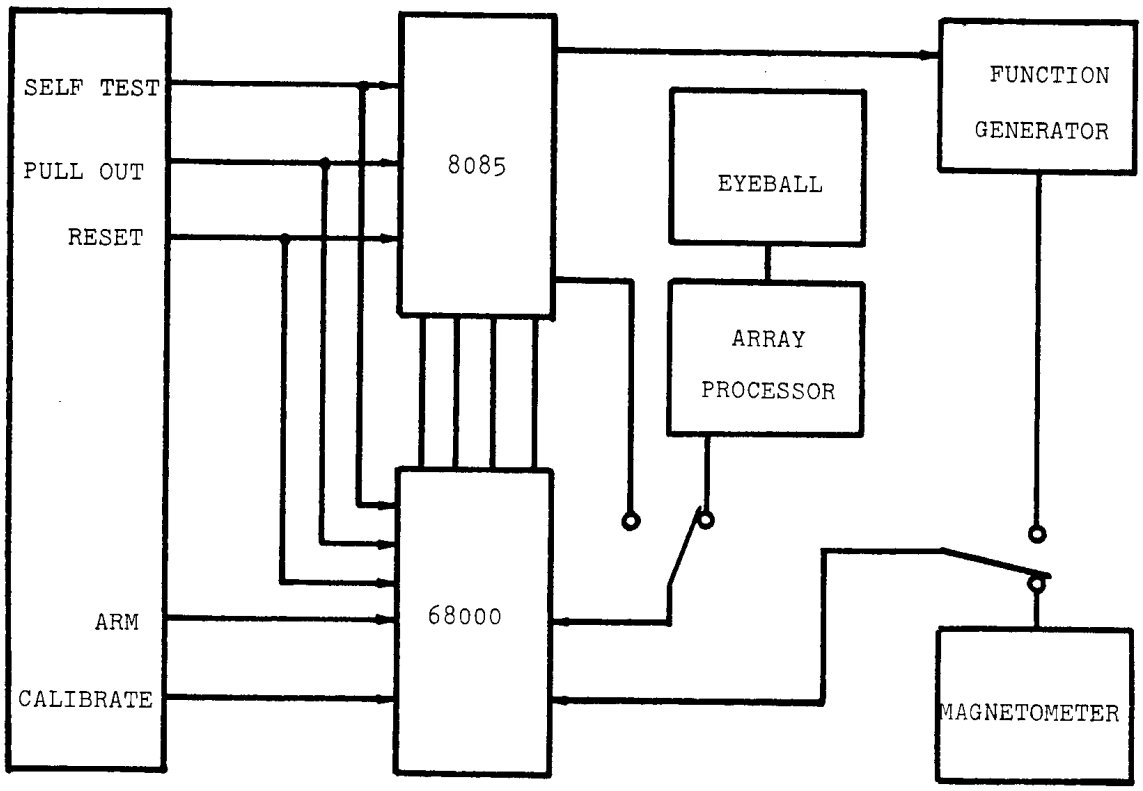


**Figure 3**

**Control Box Interface**