Space Testing of the Advanced Instrument Controller

***Todd Goforth, +Scott R. Cannon, ~James Lyke**

*Maxwell Technologies, Alb., NM. goforthT@plk.af.mil

+Utah State University, Logan, UT. scott@cannon.cs.usu.edu

~Air Force Research Lab, Space Vehicles Directorate, Alb., NM. lyke@plk.af.mil

ABSTRACT. An extremely compact, low-power instrument controller and data processor system has been developed for space-based applications. Known as the Advanced Instrument Controller (AIC), this hybrid device contains both digital and analog components in a package less than 5 grams in weight and 2 x 3 cm in size. Based on the Intel 8031/51 microprocessor and implementing a superset of the 8051 instruction set, the AIC supports 128k of SRAM, 128k of EEPROM, four 8-bit parallel ports, six serial communications ports, 32 analog 12-bit A/D channels, and eight D/A channels. Rugged (30k g) with a wide operating temperature range (-120 to +80 C), the AIC supports a number of power saving modes, nominally consuming <50mW with 0.5mW in standby sleep mode. A space experiment was designed to exercise the controller in a harsh environment. Flying on the small STRV-1d satellite, a joint US and British program, the experiment will collect data on AIC operation during 600+ minute highly-elliptical orbits which will expose the experiment to high radiation levels and possibly significant solar flare events. Scheduled to fly in spring 2000, STRV-1d will be commissioned for one year.

Introduction

Over the next ten years, hundreds of new operational satellites costing billions of dollars will be inserted into orbit [1]. Each operational satellite requires that a myriad of different functions be controlled and monitored; ranging from very complex attitude or tracking functions to low complexity spacecraft temperature or spacecraft battery voltage monitoring. Often these tasks are performed by a single central computer. Because there are many different micro-processors designed for space which can provide the necessary processing power, the spacecraft designer may only need to match satellite processing needs to a particular processor's capabilities. For small satellite applications however, there are significant gaps in the spectrum of space processors available for use at the lower performance (<1MIPS), low power, and small size end. This gap often forces the designer to use high-performance or high current processors for low complexity tasks. This results in power budget problems as well as greater investment, development time, and system complexity.

By contrast in non-space applications, the plummeting cost for a fixed quanta of computational capability has driven an engineering economy which favors the inclusion of many individual processors in all but the simplest systems. Yet space systems, for the most part, continue to employ a limited number of processors and seem to follow a centralized model. The reason for this difference is not clear, but it is reasonable to suggest that the size, weight, and power of many processor boxes is a factor. Because small satellite systems have significant dimensional, power, and weight constraints; reducing flight hardware is a prudent engineering goal.

Distributed processing is an attractive design model, since the wiring complexity of the overall spacecraft harness can often be substantially reduced by localizing processing at the point of use. Furthermore, the ability to delegate subsystem processing makes the engineering of complex, real-time systems more tractable. In this case, each co-located processor is responsible for managing only one set of real-time interactions. This effect is not unlike introducing a "temporal slip-ring" to the central processor, greatly alleviating an otherwise complex, interwoven set of real-time interactions with subsystems. If the size, weight, power consumption, and cost of an embeded processor could be reduced, it would be possible to more completely realize the potential benefits of distributed processing. Distribution of processing also reduces the burden faced by any particular processor. If the system designer could reduce the number of overall

tasks that a central spacecraft computer was required to do by using lower performance processors in a distributed processing scheme, the overall development cost would also be reduced.

A practical distributed processing scheme would involve using many low cost, low power and low complexity processors, distributed over the entire satellite. These processors could be used to handle low-level tasks such as controlling subsystems and monitoring the status of various sensors. Each Remote Processor Unit (RPU) would provide processing and instrumentation services for various satellite functions, and would be linked to the central microprocessor via a simple serial or parallel data path. The RPU's could be used to control, monitor, and process satellite health and status data such as attitude, voltage and current levels, temperatures, solar panel position, etc. The RPU's could also request service from the central microprocessor for a subsystem if a condition warranted an action on its part.

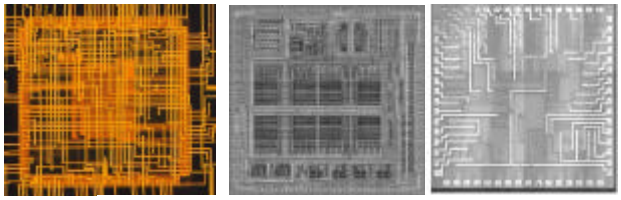
This paper addresses the issue of using flexible but lower performance processors in a distributed processing scheme by examining the suitability of a relatively new micro-controller, the Advanced Instrument Controller (AIC) Multi-Chip Module (MCM), for use as an RPU. The AIC approximates the "system-on-a-chip" model through the synergistic combination of integrated circuit and advanced packaging technology. The AIC, by virtue of its extremely small size, weight, and power consumption, can readily support the paradigm for pervasively distributed processing in spacecraft. In order to address the obviously wide range of applications that such an integrated miniature computer could support, it is important to examine how AIC operates in harsh environments.

The AIC was originally developed for the NASA New Millenium Program's Deep Space II (DS2) mission. Though this Mars mission is a physically demanding test, the radiation exposure that the AICs will experience in that mission is very low. One section of this paper will discuss the use of the AIC in a more stressing radiation environment. In this case, AIC was applied as the core of a Low-Power Electronics (LPE) experiment. The details of the design, development, test, and qualification process of this LPE experiment will be described. As a consequence of this and other applications, it was also necessary to develop software to provide a better support environment conducive to software development and testing. A common dilemma of system-on-a-chip modules is the lack of commodity in-circuit emulation facilities. A creative solution in the form of extended resident debug monitor was established to support more fluidly the development of embedded software. The design of this software is also described in detail.

AIC MCM Design

The AIC is an embedded, mixed signal MCM that possesses features to address the issue of distributed processing in an enabling way. These features include: the ability for stand-alone and self-contained operation, a rich collection of built-in analog and digital instrumentation control and sampling capabilities, a large number of communications ports, and an in-system non-volatile program and data storage facility. Because the unit is miniaturized, low-power, ruggedized, and self-contained, it is easily embeddable in almost any point location and is an ideal candidate to support the wide-range of low-level processing requirements that exist in complex electronic systems. The novel design of the AIC MCM consists of a digital micro-controller ASIC (Applications Specific Integrated Circuit), an analog ASIC, and a resistor ASIC [2,3].

Figure 1 illustrates some of the components contained in the AIC MCM, which include a) a 0.35 micron, 3.3V CMOS technology central processing unit, b) two commercial memory devices (Hitachi 128kx8 SRAM and EEPROM), and c) a 70,000 device analog ASIC built in 2 micron CMOS (Orbit) process.



(a) (b) (c)

Figure 1. Some of the ICs used in the AIC. (a) AIC51 central processing unit, shown here during an intermediate fabrication step (HDI traces visible over die and bond pads). (b) Analog ASIC. (c) Resistor ASIC.

The AIC MCM is packaged in a version of the High-Density Interconnect (HDI) MCM process that employs a plastic substrate [4]. A simplified view of the MCM fabrication sequence is shown in Figure 2. Fabrication of plastic MCMs in the HDI process begins with a thin (1 mil) Kapton polyimide sheet (Fig. 2a). The internal ICs described are introduced through placement (Fig. 2b) onto the Kapton sheet through a thermoplastic adhesive interface. The plaskon substrate

of the MCM (Fig. 2c) is poured or molded over the placed components. Next, the standard formation of a multilayer copper-Kapton interconnect system creates a dense wiring manifold (Fig. 2d) between the contacts of ICs, while also forming the conductor land areas for surface mount component (Fig. 2e) and leadframe (Fig. 2f) contacts.

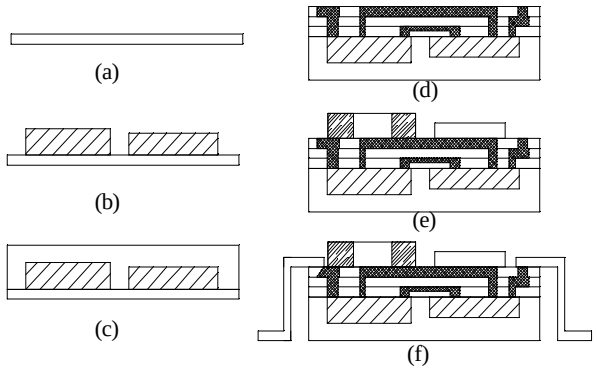


Figure 2. AIC MCM fabrication sequence.

The form of MCM technology used in the AIC, referred to as *plastic* HDI, has several important advantages, some of which were particularly important to its original application as an interplanetary probe controller. First, the patterned overlay technology allows for tightly coupling IC components. The custom ASICs took advantage of this situation by employing smaller pad driver cells for bond pads, due to the reduced capacitance of intra-modular wiring. This design approach permitted lower propagation delay and power consumption. Second, the MCM design approach provided for superior impedance control, particularly for the analog portions of the design. Signal integrity models developed for the AIC indicated that noise coupling onto signal lines going into the A/D convertor were well below one quarter of the signal level corresponding to a least significant bit. The HDI construction provided for minimum size and weight. Here, components were mounted both within the substrate and onto its surface. The use of plastic instead of ceramic resulted in a 50% weight reduction and undoubtedly contributed to its extremely rugged design.

Although the AIC was not specifically designed to be radiation tolerant, ground testing has shown that device is still suitable for most LEO applications [5]. In light of this limitation, development of radiation hardened versions of the analog ASIC and the central processing unit are currently being funded by two different USAF program offices [3].

A simplified block diagram of the AIC architecture is shown in Figure 3, and a digitized picture of an AIC MCM is shown in figure 4. The AIC was designed to be a versatile, easy to use tool, with many features and functions which allow it operate as a stand-alone system.

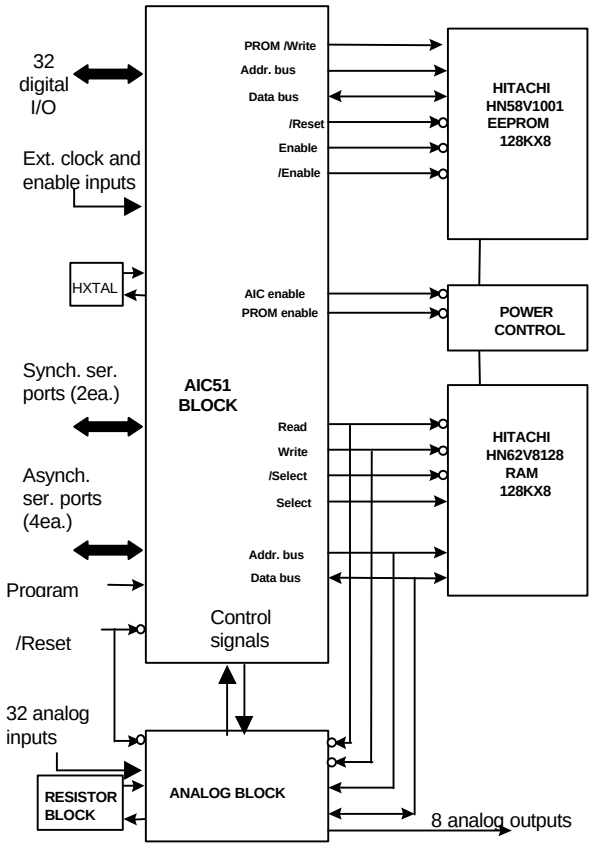


Figure 3. AIC function block diagram

The AIC is based upon the Intel 80C51FC micro-controller instruction set and is binary compatible with that family of processors. In addition, the latency of the MOVX (move using external memory) instruction has been significantly reduced. Added to the instruction set is a special write-to-EEPROM instruction (MOVN).

The operating speed is 11/n MHz (n = 1, 2, ...) from a built-in clock. External memory consists of 128k x 8 EEPROM and 128k x 8 SRAM (64k program, 64k data). The miniaturized AIC is only 3.22cm x 2.0cm x 0.3cm and weights less than 5 grams. On reset, 64k of program is automatically copied from EEPROM into RAM -- execution then proceeds from RAM.

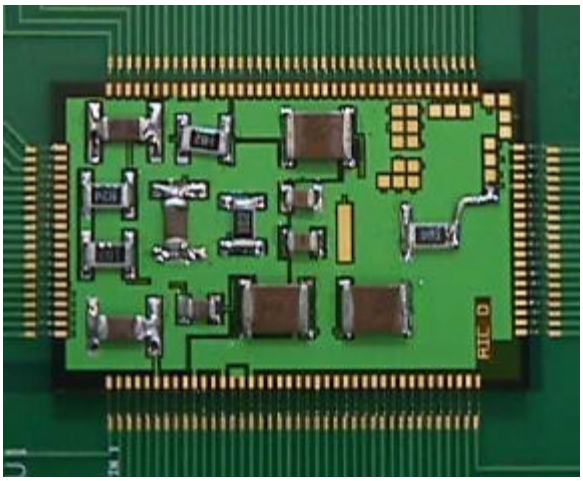


Figure 4. AIC MCM Wire-Bonded Test Board.

AIC MCM peripheral features consist of the following:

- 32 digital I/O pins; 24 bi-directional, 8 output only (4 default high, 4 default low.
- Two 16-bit timer/event counters (one timer contains a powerful capture/compare module).
- Six serial ports; four RS-232 format at CMOS levels (asynchronous) and two half-duplex synchronous.
- A 12-bit A/D converter with four operating modes, 32 multiplexed input channels (0 - 4 volts), a 40 usec conversion time, and a 15 KHz conversion rate.
- Eight channels of D/A conversion (each 10-bit resolution).
- Seven sources of interrrupt.

There are several important AIC special features that facilitate embedded and distributed applications:

- Power-on reset
- A built-in oscillator
- In-situ reprogrammable through serial port 0.
- Two operation modes; nominal (clock rate choices of 11, 5, 2.5, or 2 MHz) and a standby or lower power mode (clock rate 200 Hz).
- Low power consumption: 50 mW typical at 5 MHz, <1 mW during low-power mode.
- Fully static design.
- Power supply; 5 volts (+/- 5%) and 3.3 volts (+/- 5%).
- Highly rugged design; 30k G forces with an operating temperature range of -120 C to +80 C.

The LPE Experiment

Recent work with the US/UK Space Test Research Vehicle (STRV) series of satellites has created the opportunity for the demonstration of space experiments in harsh radiation environments. The STRV-1d satellite, which will be launched with the STRV-1c companion spacecraft, will be placed into highly elliptical orbits where they will pass in and out of the Van Allen radiation belts every 10.5 hours during their one-year mission [6].

Because of the obvious power and weight savings of the AIC MCM over conventional data acquisition and microcontroller circuits, the successful demonstration of the AIC MCM in a space environment will benefit many future space projects. This demonstration will also prove the viability of the AIC MCM as a candidate for remote processing applications.

The Low-Power Electronics (LPE) experiment (Figures 5 and 6) was designed to demonstrate circuits 10 - 100 times more power efficient than are currently used, and in particular examines (1) reduction in feature size, and (2) digital/analog integrated co-design and how these systems might perform in a synergistic environment. The main focus of the experiment is the AIC MCM. (A more detailed description of the complete LPE experiment is presented elsewhere [7]).

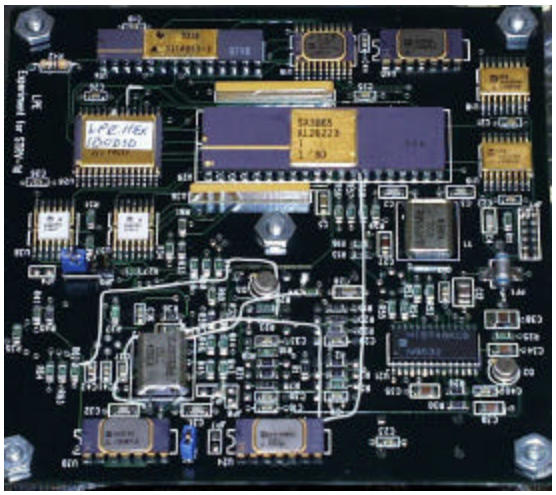


Figure 5. Top View of the LPE Experiment Board.

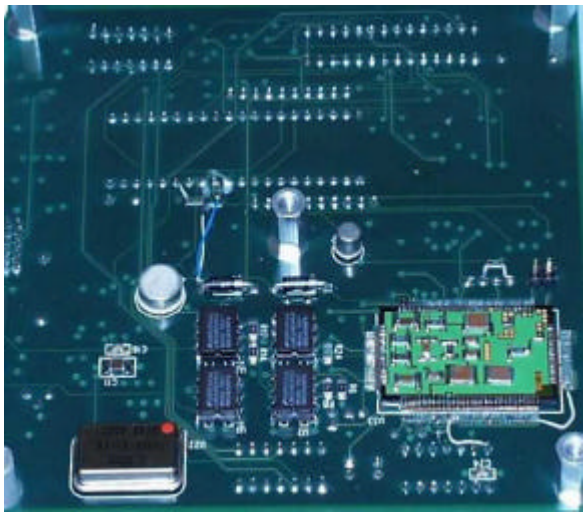


Figure 6. Bottom View of the LPE Experiment Board.

LPE Experiment Description

The LPE experiment board is designed to evaluate the performance of two low-power integrated circuits in a space environment. One of these integrated circuits, the AIC MCM, is a custom component available through the Air Force Research Laboratory.*[1] The criteria used to determine the success of the experiment will be based largely on the duration of proper operation of the AIC MCM.

The LPE experiment board will functionally test the AIC MCM by applying appropriate stimulus inputs and then collecting the response data. Once collected, the data is transferred to the main spacecraft computer via a serial communications link, and then transmitted to a ground station for evaluation. The following subsections give brief descriptions of the several LPE experiment board functions. In addition to processor operation, the AIC MCM functions which are monitored by the LPE experiment board are;

- Power Dissipation in Active and Standby modes,
- ADC Zero Offset Error,
- ADC Full-Scale Error,
- ADC Linearity,
- DAC Zero Offset Error,
- DAC Full-Scale Error,
- DAC Linearity,
- Self-Test Mode Initialization and Execution,
- On-Chip RAM Performance and Integrity,
- On-Chip ROM Performance and Integrity,
- I/O Port Functionality,
- Synchronous Serial Port Functionality, and
- Asynchronous Serial Port Functionality.

The LPE experiment functions are controlled by a separate on-board SA3865 radiation hardened 8-bit microcontroller. The SA3865 is functionally equivalent to the Intel 80C51BX micro-controller which simplified interfacing to the AIC. The SA3865 controls all of the AIC experiment functions and timing, and collects data from the AIC MCM. The SA3865 is periodically commanded by the spacecraft data handling computer to exercise the AIC MCM and then interrogate the device to collect experiment data packets.

This data consists of 143 bytes from the AIC MCM per interrogation showing the status of the above bulleted functions. The SA3865 uses an 11.0592MHz crystal, and communicates with the spacecraft data handling computer system at 9600 baud through an RS-422 interface. The SA3865 communicates with the AIC via a two-wire handshake which is controlled by the AIC MCM. Data transfered to the spacecraft data handling system is stored to on-board radiation tolerant RAM for eventual ground telemetry.

AIC MCM A/D Converter Evaluation

The AIC MCM ADCs are tested in a variety of ways. Four of the ADC inputs are connected to the output of a buffer amplifier, which is driven by a radiation hardened voltage reference. Using this reference will give a clear indication of how stable a typical AIC MCM ADC channel is when used in a temperature and total dose varying environment such as space.

The other 28 analog inputs are broken up into groups of either 3 or 4, with each group being driven by one of the AIC MCM DAC outputs. Each of the DAC outputs is coupled to a group of ADC inputs through a 10K Ohm resistor. Because the DACs will be programmed to output one of four different voltage levels, it is hoped that ADC and DAC linearity, zero offset error, and full-scale error performance can be tracked as a function of temperature and total dose radiation. One important thing to point out is that if these tests show degradation, it might not be absolutely clear if the degradation occurred in the DAC or the ADC. If

*[1] Air Force Research Lab, 3550 Aberdeen Ave. SE
Kirtland AFB NM 87117-5776 (505)846-5812
Point of Contact - Mr. James Lyke

degradation does occur laboratory testing to determine the exact source of degradation may need to be carried out.

AIC MCM I/O Port Functionality Evaluation

Each of the four, 8-bit I/O ports are exercised to some degree or another. The main test function uses two bits from each port to transfer every data byte from the AIC MCM memory to the main experiment microprocessor as part of the normal data collection passes. A different I/O line is tested by serving as an interrupt input for the AIC MCM. This I/O is a dual function I/O and is tested for both its interrupt function as well as its standard I/O function. These tests provide a clear indication of I/O functionality over temperature and exposure to total dose radiation but will give no indication of I/O drive capability.

AIC MCM Asynchronous Serial Port Evaluation

The AIC MCM contains one asynchronous serial port but this single port is multiplexed to four sets of asynchronous serial port pins. Because the asynchronous serial ports will operate in full duplex mode, it was possible to evaluate the transmit and receive operations simultaneously. Asynchronous serial port operation was verified in each of the four possible modes. After each test operation is completed the receive buffer and the control register are written to internal AIC MCM RAM to be later downloaded to the SA3865 as part of a data packet.

AIC MCM Random Access Memory (RAM) Evaluation

The AIC MCM has 128K x 8 of internal random access memory, which is split into an upper block and a lower block each 64K x 8. The AIC MCM controller software uses the RAM for storage of experiment data prior to transmitting it to the SA3865 controller. The RAM is checked prior to every data collection pass by writing and reading four different patterns; 0x55, 0x00, 0xFF, and 0xAA. The test results are stored as codes in internal AIC MCM RAM to be later downloaded to the SA3865.

AIC MCM EEROM Evaluation

The AIC MCM has 128K x 8 of internal ROM which is also split into an upper block and a lower block each 64K x 8. Because the AIC MCM system software is stored in ROM, and because testing ROM requires the continual resetting of the MCM, the ROM tests that are done on-orbit utilize the Built-In Self-Test (BIST) functinality of the AIC. This built-in routine verifies system ability to read ROM. The BIST results are stored in internal AIC MCM RAM and later downloaded to the SA3865. Although the BIST is only performed on power up, the BIST result will be transmitted with every data download.

LPE Software Development

The AIC MCM software for the LPE experiment was written using a commercial off-the-shelf code development package for the Intel 80C51. The assembled hex code was then downloaded to the AIC MCM, which was soldered onto the flight board. The download was accomplished in-situ through a three-pin header which was designed into the board for ease of re-programmability, and not removed until final integration began. Two other two-pin headers, also designed into the board and removed before final integration, were used for assertion and de-assertion of */Reset* and the *Program* pins. Use of these headers allowed newly developed revisions of software to be loaded into the AIC MCM by using the Utah State University, AIC Debug Monitor (described below). This procedure required only the use of a laptop computer and an in-line voltage level shifter, which was required to mate the PC UART voltage levels to the AIC MCM UART voltage levels. Although the LPE board was not designed to take advantage of stand-alone in-situ re-programmability, the necessary functions to do this could have easily been designed into the board and handled by the main system microprocessor. Taking this concept one step further, it is possible to completely control in-situ reprogrammability on-board assuming that new software programs are stored in on-board memory.

Code Development Support

The AIC was designed for extrememly compact and low-power embedded designs. As a result, there are no external connections for in-circuit emulation probes or pods, nor are there dedicated JTAG or other emulation test ports. Code development and testing for complex software on the AIC requires additional support. Two software packages were developed during this experiment to provide an efficient and dependable code development environment.

AIC Debug Monitor

To support interactive debugging and code testing, a semi-independent Debug Monitor module (DM) was developed. The DM is intended to co-reside with an application in AIC EEPROM code memory. This

debug monitor serves as a background routine to support interactive application debugging with a host console through the AIC serial port 0. It can be controlled with a simple terminal connected to the AIC or by a host PC running the AIC debugger Graphical User Interface (described later). The intented purpose of the DM was to use a small software routine on the AIC to emulate the essential controls normally available with an in-circuit emulation probe.

The AIC Debug Monitor is compatible with the Keil 8051 development tools (C Compiler, Assembler, and Linker), although only minor modifications would be required for another commercial compiler and linker package. The DM is linked with a user application into a single HEX file to be loaded into AIC EEPROM.

Upon leaving reset, the DM initially obtains control of the AIC and initializes serial port 0 to a known baud rate and known communications settings. It then enables serial port interrupts and jumps to the beginning of the application code. While the application is running, all serial port 0 input characters are first examined by the DM. Reception of the '!' character places the AIC in debug mode and control is assumed by the DM (fig. 7).

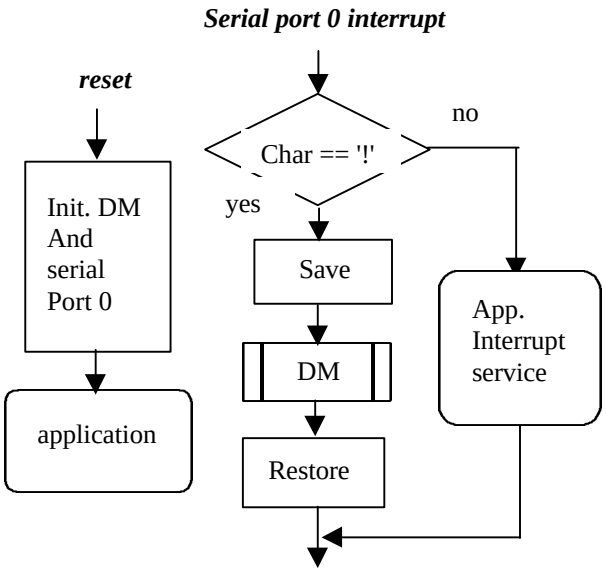


Figure 7. Debug Monitor operation

While the AIC is not in debug mode, the DM simply passes all control and all other characters to the application serial port interrupt service routine (if present) for any input character handling. All transmit interrupts are ignored by the DM and thus are available to the applications interrupt service routine. As a result, an application is essentially uneffected by the presence of the DM as long as the special '!' character is not transmitted to the AIC during operation.

When the AIC enters the DM for debug mode, the current state of the application is saved in a protected area of memory. This includes all special function registers (interrupt states, program counter, flags, etc.) that the debug monitor may modify. Clocks or timers are not stopped, but all internal and external interrupts are disabled or suspended until the application is resumed.

Debug Monitor Commands

The following commands (Table I, Table II) can be entered as text strings from a dumb terminal (or can be generated by a special graphical user interface in response to window controls as described later):

Table I: Debug Monitor Notation

xx xx …	Up to 128 2-byte hex values (00 to FF) separated with spaces.
xx	A 2-byte hex value (00 to FF).
aaaa	A 4-byte hex address (0000 to FFFF)
aa	A 2-byte hex address (00 to FF)
n	A single digit (0 to 9)

Table II: Debug Monitor Commands

Command	**Action**	**DM response**
!	Enter debug mode	<version>:
D	Read local RAM	D xx xx …:
X aaaa	Read extended RAM	X[aaaa] xx xx … :

E aaaa	Read code RAM	E[aaaa] xx xx …:
S	Read spec. function regs.	S xx xx …:
W aaaa xx xx …	Write local RAM	W[aaaa]:
M aaaa xx xx …	Write extended RAM	M[aaaa]:
A aaaa xx xx …	Write code RAM	A[aaaa]:
Z aa xx	Write spec. function reg.	Z[aa]:
Y aa x	Immediate write spec. func. Reg.	Y[aa]:
P aaaa	Change prog. counter	P[aaaa]:
B aaaa	Set breakpoint	Bn[aaaa]:
C n	Clear breakpoint	C n:

When the operator enters the 'G' or Go command in debug mode, the application state is restored and the application continues -- unless a breakpoint is reached. When a breakpoint is encountered, the AIC again enters the debug mode and the address of the breakpoint is transmitted to the host terminal or PC.

The 'Write SFR' and 'Change PC' commands do not take effect until the 'Go/Continue' command is executed. The 'Write SFR' command is used for registers A, B, DPTR, SP, and PSW. The 'Immediate write SFR' command occurs immediately and is normally used for other SFRs including ports and port control registers.

The write commands expect a string of 128 2-byte hex values, the user must provide the entire list even if only a single byte is to be written. In the case of the 'Read SFRs' command, the debugger returns 130 hex values; the first 2 values (xx xx) represent the current application program counter. The 'Write SFR' and 'Immediate write SFR' commands expect the actual SFR address.

The AIC debugger does not echo user input. In general, the AIC debugger ignores spaces and will not send <cr> or <lf> control characters. The AIC debugger is case sensitive and all ASCII hex values must be entered in upper case.

Obviously, the user is able to directly change AIC special function registers and memory upon which the application program and debugger may be depending. For example, the debugger uses some local and external ram memory for variable space. If the user overwrites these variables with new values, the debugger may be harmed. The user should carefully consider a load map listing prior to modifying memory to be sure only application variables or unused memory locations are being modified. Similarly, an immediate write to the stack pointer may harm the debugger, since it may be currently using the stack pointer register. In any case, a hardware reset on the AIC will force a fresh version of the code to be loaded into the AIC ram memory.

Breakpoints

Both static and dynamic breakpoints are available in the debug monitor. A static breakpoint can be forced in application code with a call to the BP() function macro. The argument is any positive integer. For example;

```
if (a == b)
{  x = y + b;
   BP(1);  // force breakpoint
 // pause and enter debug mode
}
```

The BP() function simply simulates the reception of a '! debug-mode character on the serial port. As such, when the application program executes the BP() function, the application will pause and the system will enter the debug mode as described above. Control is returned to the application with the 'G' command.

Dynamic breakpoints are added to the application while the AIC is in debug mode. The 'B' and 'C' commands may be utilized to set and clear up to 10 dynamic breakpoints. When the "Set Breakpoint" command is sent to the AIC in debug mode, a breakpoint is defined for the address specified. When the breakpoint is successfully registered in the AIC, the display will show the breakpoint number and address

in the form “n[aaaa]”. To clear a breakpoint, send the “Clear Breakpoint” command with the appropriate breakpoint number.

Dynamic breakpoints are effected by saving and then replacing the existing instruction at the breakpoint address with a BP() function call. Setting a breakpoint in ram-resident code has no effect on EEPROM – when the AIC is reset, a new copy of code is loaded from EEPROM and all breakpoints will be erased.

There are certain restrictions imposed by a software-implemented debug monitor regarding breakpoints. Setting dynamic breakpoints in assembly is not a trivial activity and requires some understanding of linker map files. It is the operator’s responsibility to set a breakpoint on a valid instruction address. If for example, a breakpoint is specified for the second byte of a 2-byte instruction, unpredictable operation will result. A breakpoint results in 3 code bytes being replaced in the original code. As such, the operator must avoid setting a dynamic breakpoint on a 1 or 2 byte instruction if possible. If this is not possible, one can avoid setting a breakpoint directly above a destination or labeled instruction since the destination instruction may be replaced by the breakpoint but not be in the instruction sequence reaching the breakpoint. If the application is in C, simply placing a breakpoint at the beginning of a line containing executable instructions (as indicated in the linker output file) will insure the above conditions are met. The user may not place a breakpoint at the end of a block (on a ‘}’ line).

AIC Graphical User Interface

The AIC *Graphical User Interface* (GUI) was developed to support code development and debugging on the AIC by providing a more intuitive and user-friendly interface to the Debug Monitor described above. While the GUI is not necessary for code development, its use provides the programmer with a simpler and more robust interface. The GUI runs on a PC host and generates the appropriate Debug Monitor command strings associated with a simple window display of buttons, options, and dialogs. Responses from the Debug Monitor are intepreted by the GUI and displayed (fig. 8).

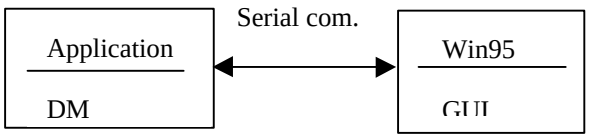


Figure 8. Development System Configuration

Data displays

Three data display window areas are presented to the user in the GUI (fig 9.). In the “Status” display area, information messages regarding the state of the system and PC interface are displayed. In the “AIC comm” display area, text sent by the AIC is displayed. In the “Data” display area, AIC memory segments are displayed. This last display area is generally editable and is useful for modifying AIC memory segments using the Read Data and Write Data control buttons.

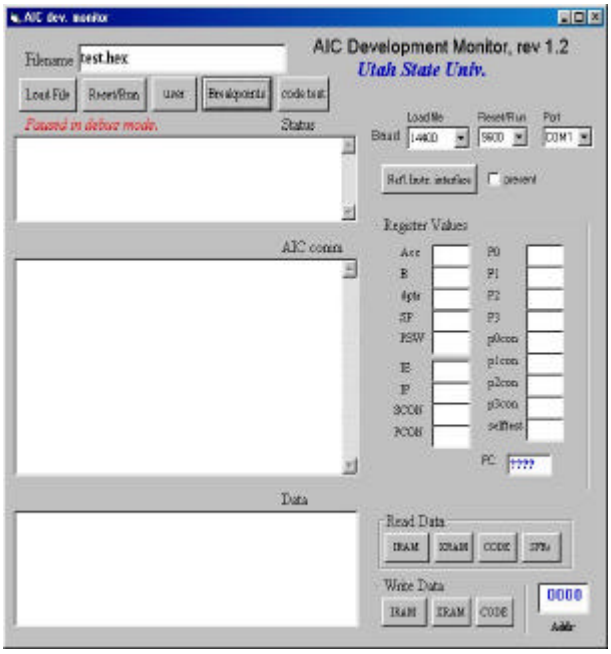


Figure 9. Graphical User Interface Window

Operation Controls

- LOAD FILE Downloads a hex file to EEPROM. The filename is first entered in the Filename text window. After downloading a program, this button becomes inactive and unavailable. While the program is being downloaded, the message “downloading” is presented in the status window followed by “download complete” when EEPROM has been loaded. This may take several minutes – dot progress markers are displayed to show the is active.
- RESET/RUN Reset the AIC and run the application program currently

in EEPROM. While running, the application program communications from the AIC appear in the AIC comm display. When a reset command is successful, the message "AIC reset" is displayed in the status window.

- DEBUG Interrupt the application program on the AIC and enter the debug mode. The Register Values, Read Data, and Write Data button groups are only available while in the debug mode. After entering the debug mode, this button changes to "User" (see below).

- RUN Leave the debug mode and continue with the suspended application program.

- BREAKPOINTS Show the breakpoint control form (see below)

Read Data controls

- IRAM Read the lower 128 bytes of local ram from the AIC and display in the Data display area.

- XRAM Read 128 bytes of extended ram block 1 (upper 64k) beginning at the address in the "Addr" window into the Data display area. The Addr window should contain a 4-character hex address in the range 0000 to FF80.

- CODE Read 128 bytes of code from extended ram block 0 (lower 64k) beginning at the address in the "Addr" window into the Data display area. The Addr window should contain a 4-character hex address in the range 0000 to FF80.

- SFRs Read 128 bytes of special function registers from the upper 128 bytes of local ram and display in the Data display area and into the Register Values display group. The Addr window should contain a 4-character hex address in the range 0000 to FF80.

Write Data controls

- IRAM Write the bytes currently in the Data display area into the AIC local ram at the address in the Addr window. The Addr window should contain a 4-character hex address in the range 0000 to FF80.

- XRAM Write the bytes currently in the Data display area into the AIC extended ram block 1 (upper 64k) beginning at the address in the "Addr" window. The Addr window should contain a 4-character hex address in the range 0000 to FF80.

- CODE Write the bytes currently in the Data display area into the AIC extended ram code ram (lower 64k) beginning at the address in the "Addr" window. The Addr window should contain a 4-character hex address in the range 0000 to FF80.

Register Values controls

When the Read Data control button "SFRs" is clicked, these display windows are filled according to the special function registers read from the AIC. Most of these windows are editable. Hitting the RETURN keyboard key in one of these register display windows causes the current value in the display to be written back to the associated SFR in the AIC.

The effect of changing program control registers may not be immediate. For example, changing the PC, A, B, and dptr registers cannot take effect until the application program is resumed (see the User operation command above). The effect of changing any of the port or port control registers is immediate.

Breakpoint control form

The Breakpoint control form (fig. 10) is used to enter *dynamic* breakpoints into the ram-resident application program on the AIC. Up to 10 breakpoints may be specified. When the "Set Breakpoint" button is selected, a breakpoint is defined for the address in the "Address" window. This window should contain a 4-character address in the range 0001 – FFFE.

When the breakpoint is successfully registered in the AIC, the display list will show the breakpoint number and address in the form "n[aaaa]". To clear a breakpoint, select or highlight the breakpoint in the display list and select the "Clear Breakpoint" button. If no breakpoint is selected, this button will clear the top breakpoint by default.

When the application program reaches a breakpoint, the program is paused and the AIC enters the debug mode. Setting a breakpoint in ram-resident code has no effect on EEPROM – when the AIC is reset, a new

copy of code is loaded from EEPROM and all breakpoints will be erased. To hide the breakpoint control form and return to the main control form, the user selects the "Done" button.

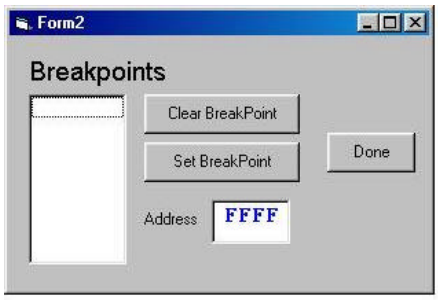


Figure 10. Breakpoint Control Form

Summary

Even as the commercial industry (e.g., laptop computers) continue to exploit advances in low-power microelectronics and distributed processing, this venue of research is particularly enabling in space systems, where every required joule of energy comes with a price tag, and costs associated with central processor development is very high. The advancement of electronics for space requires examination of these new trends, both in terms of technology development and in testing/verification of the space worthiness of these technologies.

The AIC promises to be a very practical instrument controller for small satellite applications where distributed processing can be utilized, particularly those satellites requiring both digital and analog controls and sensors. The Debug Monitor and Graphical User Interface software developed to support the AIC provide a practical and efficient code development environment for applications implementation, debugging, and testing. The success of the STRV-1d test satellite will provide valuable data on the robust nature of this device.

Since the delivery of the AIC MCM to Deep Space II, other AIC MCM demonstration systems have been provided to several NASA centers to enable quick prototyping and evaluation of the unit. The Marshall, Lewis, and Johnson space flight centers are planning missions using the AIC MCM, and JPL is looking into using the AIC MCM for the Mars 2003 mission [3]. The U.S. Air Force is considering the AIC MCM for new applications such as satellite attitude control and process control [3].

The AIC MCM is also being considered for several terrestrial distributed processing applications. Contracts are currently in place with the U.S. Navy and the Joint Strike Fighter program, which envision using the AIC MCM in a condition-based maintenance application [3].

Acknowledgments

Acknowledgment is given to the Boeing Corp. (Jay Clement, Scott Davis), Mission Research Corp. (Dave Alexander, Robert Turfler, Mike Williams), and Utah State University for providing technical support during the development of the LPE space experiment hardware and software.

References

[1] Troy Thrash, "Little LEOs: Bigger Isn't Always Better", *Launchspace*, pp. 46, 47, January/February, 1999.

[2] James Lyke and Curt Jingle, "AIC/DS2 Users Guide", Phillips Laboratory technical report, 1998.

[3] James Lyke and David Alexander, "Design of an Ultra-Miniature Control Processor for Space Application", *1999 GOMAC Digest of Papers*, pp. 256-259.

[4] James Lyke and Robert Wojnarowski et al., "Three Dimensional Patterned Overlay High Density Interconnect (HDI) Technology", *Journal of Microelectronic Systems Integration*, **1**(35), July 1993.

[5] William X. Culpepper et al., "Radiation Test Summary Report For The Experiment Controller For The Mars In-Situ Propellant Production Precursor (MIP)", *Document Number JSC 28504*, November 1998.

[6] Wells, Nigel. "STRV-1c & 1d Mission Definition Specification", Phillips Laboratory technical report, July 1997.

[7] Todd W. Goforth, "Design and Development of a Low-Power Electronics (LPE) Space Experiment", *1999 GOMAC Digest of Papers*, pp. 320-323.