Robotics Synchronization And Information Distribution System (RSAIDS)

Gary R. Boucher
Associate Professor
gboucher@pilot.lsus.edu

Alfered L. McKinney
Professor
amckinney@pilot.lsus.edu

College of Sciences
Louisiana State University in Shreveport
Shreveport, LA 71115

Reza Sanati Mehrizy
Associate Professor
sanatire@uvsc.edu

Afsaneh Minaie
Associate Professor
minaieaf@uvsc.edu

School of Computer Science and Engineering
Utah Valley State College
Orem, Utah 84058

Abstract

Since our schools do not offer an engineering program, we teach robotic technology within our computer science curriculum. In the process of teaching robotics technology to students at a graduate or undergraduate level, it becomes necessary to synchronize more than one robotic arm for the purpose of demonstrating the interaction between robots commonly found in industrial settings. There are several approaches to doing this. The simplest approach is to connect the two machines with hard-wiring. This requires the operator to connect outputs of one machine to the inputs of another. Perhaps the reverse will be also necessary in connecting the outputs of the second robotic controller to the first.

Another method of synchronization involves the use of expensive industrial quality programmed controllers using ladder logic to evoke responses from the affected robots based on certain inputs. Most Computer Science and Computer Technology students are not familiar with such controllers but do possess a well based knowledge of several computer languages.

The third approach and the topic for the RSAIDS approach is to use a microprocessor to control the synchronization of the robotic arms. The main problem with this third approach is the fact that microcontrollers or microprocessors such as the MC68HC11 series used in the RSAIDS are difficult to program in assembly language without prior experience.

The RSAIDS which we have developed has an assembly language program that translates signals between the robotic controllers and a "Host" computer. The RSAIDS is capable of synchronizing several robotic arms using a single “Host” computer. All that is necessary for communications between the host and the separate controllers is the knowledge of a primitive set of communications instructions understood by the RSAIDS unit. This unit has been designed and constructed for around $100.00. This paper elaborates the design, construction, and application of RSAIDS in details including hardware and software requirements in details.

Communications with Robot Arm Controllers

There are two main methods for input and output of information to most robotic controllers. There exists a serial channel associated with the controllers, which allows data input and output, to and from a host computer[2]. This can be a viable alternative to communication and synchronization with these systems. However, this requires a substantial amount of programming by the student to achieve synchronized operation of multiple robots. The student also can use the simple ON/OFF digital input and output interface via the front terminal connectors found on the controllers[3]. These connectors represent the most widely used approach where the student can make direct electrical connection between the controller devices.

In the robotics lab students use the Advanced Control Language (ACL) provided by the robot vendor to write robotic control programs[2]. This language has a built-in set of I/O functions that allow the setting and clearing of output bits and the reading of input bits. Input information can be tested in IF statements to branch or hang at one certain location in the program until the bit is either set or cleared to allow the program to proceed to the next step. In short, bits can be set and tested under ACL program control in an easy and straightforward manner.

The Purpose of RSAIDS

Wiring two or more controllers together using cabling often meets with limited success as the student must master both the control language along with the proper methods of wiring the controllers. This problem of wiring is multiplied when over two controllers are to function in synchronization.

RSAIDS is an approach to greatly facilitate connections of multiple controllers where the combined operation of the robots can be brought to a simple software level using an easy to learn control language. To do this requires a small degree of both rudimentary hardware and assembly language microcontroller software.

RSAIDS represents a model, which can be modified as needed to meet these minimal requirements. Sample software and hardware interfacing schematics are provided in Appendix A as a final design, or a base for further development.

RSAIDS Communications with Host

The hardware for support of the RSAIDS software functions can be virtually any of the readily available prototype board systems that support the Motorola MC86HC11 family of microcontrollers. Most of the industry’s available board systems for these widely utilized MCUs have built-in RS-232C ports. Also, these board systems have most, if not all of their pins made available for interface to user-developed I/O devices.

Schematics are provided in Appendix A showing a suggested approach to both input and output interfacing. This interface hardware is all that is needed to connect the basic MCU to the robot controller logic. Both of these circuits can be implemented via a printed circuit board or using wire wrap technology.

The RSAIDS built-in RS-232C serial interface is connected directly, or with the use of a null modem, to the host computer's serial adapter usually on COM1 or COM2. Most computers come with this capability via a male DB-9 connector on the back panel. Some newer computers without built-in RS-232C capability may require a Universal Serial Bus (USB) adapter to convert to the serial protocol.

The RSAIDS unit is setup to communicate at a fixed 9600 Baud. This can be changed, but only by those experienced in communications with the Motorola BUFFALO Monitor system found in the MC68HC711E9 processor chip on the RSAIDS[4]. Faster speeds should consider the serial cable length and the possibility of data corruption seen with fast baud rates and long cabling[1]. The protocol for transmission via the RSAIDS is 8-bit, no parity, with one stop bit. This is standard for most modern communication applications.

Hardware Connection to Controllers

Although numerous configurations may exist for interfacing the RSAIDS unit to the robot controllers, the prototype used two inexpensive and readily available DB-25 connectors to carry the I/O logic lines to and from the enclosure housing the unit. Each DB-25 can be the connection point for several multiple-wire cables going to the robot controllers. It is suggested that these cables use breakout boxes at their ends to facilitate connection to the controllers.

Pins are provided on the controllers for screw terminal connection. It should be noted that active low input/output is present on all controllers. Thus, if the controllers input is 0 Volts the input is considered logic HIGH. For 5 Volts the input is considered logic LOW. This is taken into consideration in the RSAIDS software and reversed so that logic (1) in the ACL software at the controller site is a logic (1) at the Host.

Power Connection to the RSAIDS

The RSAIDS prototype unit runs on 9 Volts DC. This power is supplied by a standard plug in power module as used by many modern electronic devices. There is no power switch on this RSAIDS unit. If re-initialization is required, the operator simply unplugs the power from the 9 Volt wall power adapter cord where it goes into the power plug on the RSAIDS then reinsert the cord into the RSAIDS a second or two later. Although this is somewhat more inconvenient than using a reset button, there are few times that reset of the RSAIDS has been required.

RSAIDS Communication Commands

The command interface between the RSAIDS unit and the host computer is simple and straightforward. When the RSAIDS unit is powered up, internal software initializes the serial communications port of the device. This initialization requires the issuance of a single Carriage RETURN <CR> character from the host. Therefore the RSAIDS will not be in communications mode before this character is received. It is thus the responsibility of the host to provide this character after the power-up of the RSAIDS.

After power-up and initialization with the <CR> character, the RSAIDS is ready for use. There are four types of commands that can be sent from the host to the RSAIDS. These are listed below:

<u>Write Single Output</u> (Shown by Example)

WB2H<CR> This <u>W</u>rites from the Host to controller <u>B</u>'s input <u>2</u> a logic <u>H</u>igh.

WA3L<CR> This Writes from the Host to controller A's input 3 a logic Low.

If the above commands are of the correct syntax they will be echoed back including the <CR> at the end. If not accepted, the echo back to the host computer will be E<CR> for Error.

Read Single Input (Shown by Example)

RC1<CR> This is a command to Read output 5 from controller C. (*)

RA3<CR> This requests a Read of output 7 from controller A. (*)

If the command was accepted the return will be a X<CR> where "X" is either a "0" or a "1". If the command was not accepted the return will be E<CR> .

Write-All (Shown by Example)

W-1011001001011101<CR> - This writes all outputs at the same time. The first "1" following the "-" is for controller A's input 1. The next bit, a "0" goes to controller A's input 2. The fifth bit from the "-" is a "0" which sends a Logic Low to controller B's input number 1. The right most bit, a "1" writes a logic High to controller D's input bit number 4. After each Write-All the entire string of 18 characters is echoed back to the controller followed by a <CR>. If there is an error only E<CR> will be echoed back to the host.

Read-All (Shown by Example)

R-<CR> This command returns the following string to the host computer: R-XXXXXXXXXXXXXXXX<CR>. The first "X" represents controller A's output 5. The second "X" represents output 6 from controller A. Since there are only three inputs to the host from each controller the 4th, 8th, 12th, and 16th "X" has no meaning and must be ignored. Later modifications to the RSAIDS could enable a 4th controller-output (host-input) if more connection wiring is provided beyond what exists currently. In the event that the syntax was incorrect the standard E<CR> will be returned to the host computer.

(*) Note: The first four outputs of each controller are relay type outputs and are numbered 1 through 4. These are reserved and not connected to the RSAIDS. Rather, controller output 5, 6, and 7 are used to return information to the RSAIDS. The software and thus commands provided with the RSAIDS unit consider these controller-outputs (host-inputs) to be numbers 1, 2, and 3. Thus the operator must be aware that controller output number 5 is actually seen as input number 1 to the host.

There is one type of command used to send controller output information to the host without any type of prompting. The RSAIDS looks at the outputs from the controller constantly. If there is any change in any controller output the RSAIDS sends a response to the host. This response is in the form of the serial string I-XXXNXXXNXXXNXXXN<CR> , which is received by the host. Thus, each time one or more controller outputs change, the host sees the change without polling the RSAIDS. If a certain controller output goes High and then Low the I-type string will be sent twice to the host; once when the line goes High and the other when the line falls Low.

The "X" characters represent controller output A1 through D3. The "N" characters represent "Don't Care" values because as stated earlier, they are
not connected to the RSAIDS. Event driven software takes this I-type ("I" is for Interrupt) command and does what is necessary when the change occurs and nothing when there is no host input change.

Design Factors for Host Computer Software

Much care must be taken on the part of the student endeavoring to write robotic control software. If there are 4 robots to be interfaced there will have to be 5 programs in all to synchronize the operation desired. Four will be ACL language programs and the fifth will be the host computer program. The host program requirements are flexible and may be written in numerous languages including Visual BASIC, Quick BASIC, Pascal, C, C++, or Assembly language. It is the desire of the authors to see a universal package written, perhaps in Visual BASIC that will provide a simple visual interface for the user.

Considerations for such software either as a platform or just a simple program created by the user must include several major considerations. First, Write Commands require as long as 20mS from the time that a command is sent to the RSAIDS unit before the actual lines are pulled high or low on the controllers. Returning information such as the command itself that was issued or data from controller outputs, error messages etc. must be input into the host language to obtain the full features of the RSAIDS unit. This too takes up to 20mS. Each character send serially takes about 1.2mS on average to go to/from the host computer.

Changes in the outputs from the controllers issue a command in the form of I-XXXNXXXNXXXNXXXN<CR> , as explained above. This data string will require capture and interpretation to analyze what bit or bits have changed. For this type of capture an "Event Driven" software will prove to be far superior to languages that have to "Poll" the host input status with the R-<CR> command.

Commands can be given too fast from a fast host. For example; if you need a low-to-high-to-low pulse sent to input number 1 on a certain controller. Data have been lost in this process because the host language sent the commands too fast for the RSAIDS to capture them serially and respond. Delay routines are needed in some cases to allow the programmer to first send the command to go High and wait for a prescribed number of milli-seconds before the line is pulled Low again. Likewise, the controller can output data high and then low fast enough that the RSAIDS can not catch either or both of the transitions. Considerations vary from one host computer to another and from one type of host software to another.

Perhaps in the future an enterprising student will obtain a "Special Problems" credit for designing a host software user-friendly platform for this application.

Results and Conclusions

The capability of the RSAIDS unit has been demonstrated with the use of multiple SCORBOT-ER Vplus robot arms and a controller operated turntable. Microsoft Visual Basic was used to send and receive commands to the RSAIDS unit. This proved viable, but the conclusion of both faculty and senior-level student programmers was that any viable software program capable of sending and receiving via the RS-232C serial port should be usable. The fact that Visual Basic is an event-driven language allows for a great deal of latitude in asynchronously generated events.

From the above facts and personal experience with the use of the RSAIDS unit it was judged by the investigators to be a successful tool for student application to real-time robotics control.

Availability of Technical Information

The authors possess assembly language listings files and other technical materials, which can be disseminated freely to those interested in development of their own RSAIDS or similar unit. The authors welcome inquiries and requests.

Bibliography

[1] Motorola Inc., MC68HC11 Reference Manual, Motorola Inc., 1991.

[2] Eshed Robotec, Ltd., ACL Advanced Control Language Reference Guide for Controller-A 4th Edition, January 1995.

[3] Eshed Robotec, Ltd., SCORBOT-ER Vplus User's Manual 3rd Edition, pp. 3-5 through 3-10, February 1996.

[4] Barry B. Brey, Microprocessors and Peripherals Hardware, Software, Interfacing, and Applications - Second Edition, Merrill Publishing Company, Columbus, Ohio, page 272, 1988.

Biography

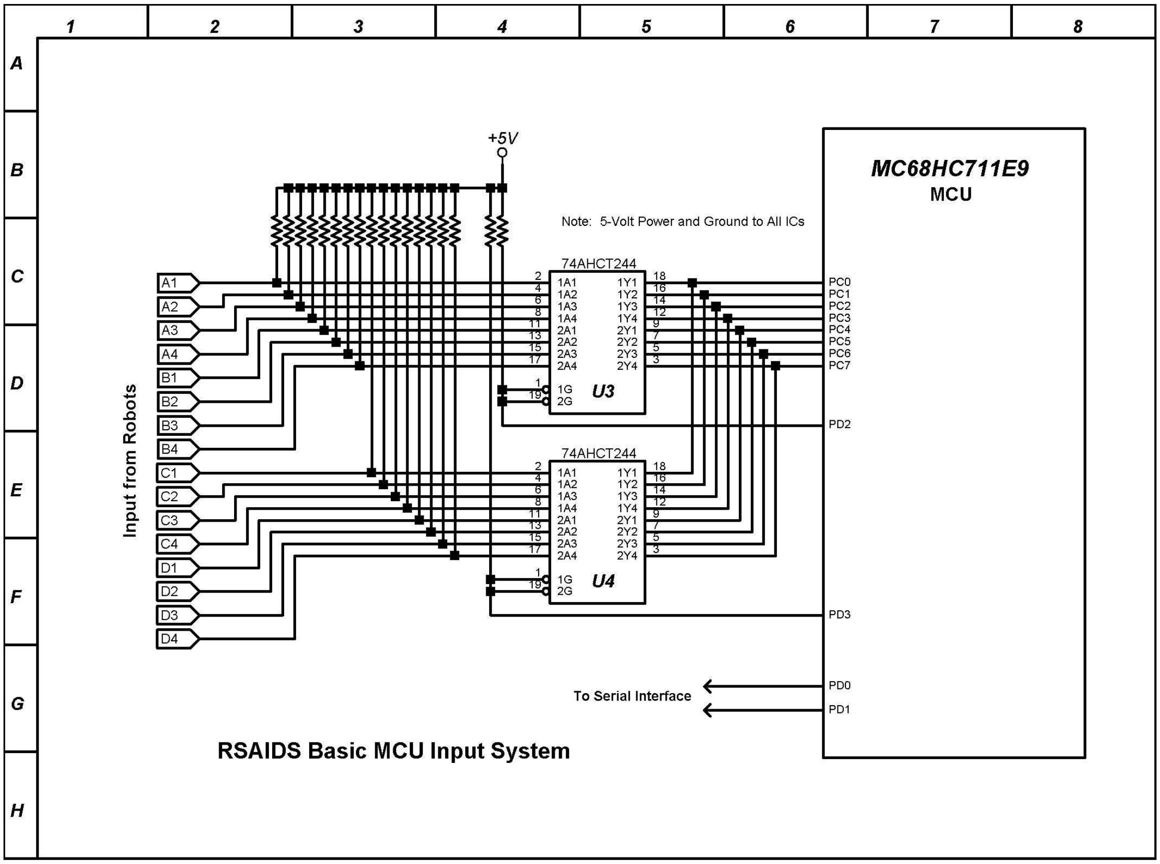
Reza Sanati_Mehrizy is an associate professor of the Computing and Networking Sciences Dept. at Utah Valley State College, Orem, Utah. He received his MS and PhD in Computer Science from University of Oklahoma, Norman, Oklahoma. His research focuses on diverse areas such as: Database Design, Data Structures, Artificial Intelligence, Robotics, and Computer Integrated Manufacturing.

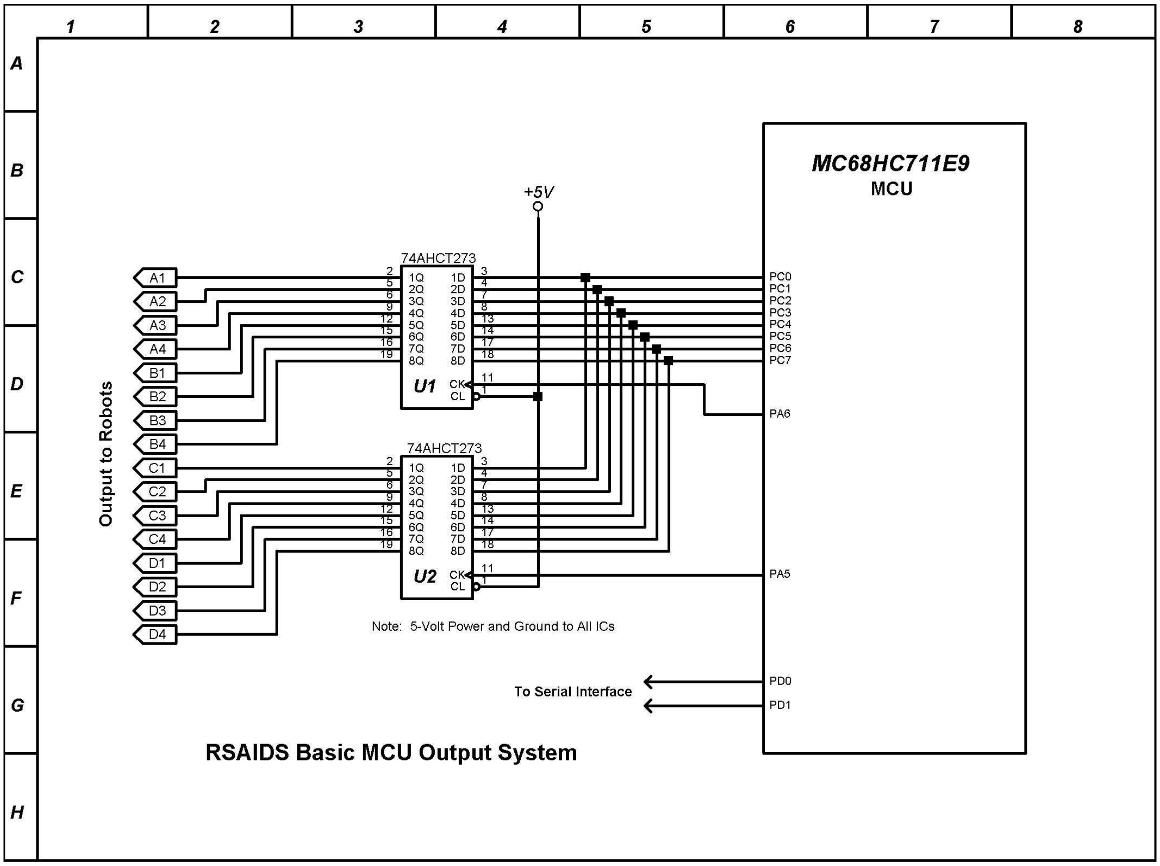
Afsanaeh Minaie is an assistant professor in the Computing and Networking Sciences Department at Utah Valley State College. She received a B.S., M.S., and Ph.D. all in Electrical Engineering from University of Oklahoma in 1981, 1984 and 1989 respectively. Her current interests are in computer architecture, digital design, and computer interfacing.

Alfred L. McKinney is professor of Computer Science at Centenary College of Louisiana**.** He received a B.S. in 1959 and a M.S. in 1961 from Louisiana Tech University and a Ph.D. in Mathematics from University of Oklahoma in 1972.

Gary R. Boucher is an associate professor of physics at Louisiana State University in Shreveport. He received a B.S., M.S., and Ph.D. in Engineering from Louisiana Tech University.

Appendix A


1
2
3
4
5
6
7
8
A
B
C
D
E
F
G
H
+5V
MC68HC711E9
MCU
Note: 5-Volt Power and Ground to All ICs
74AHCT244
U3
74AHCT244
U4
1A1
1A2
1A3
1A4
2A1
2A2
2A3
2A4
1Y1
1Y2
1Y3
1Y4
2Y1
2Y2
2Y3
2Y4
1G
2G
Input from Robots
A1
A2
A3
A4
B1
B2
B3
B4
C1
C2
C3
C4
D1
D2
D3
D4
PC0
PC1
PC2
PC3
PC4
PC5
PC6
PC7
PD2
PD3
PD0
PD1
To Serial Interface
RSAIDS Basic MCU Input System



RSAIDS Basic MCU Output System

```
* RSAIDS.ASM
*
            OPT     c
*
* Equates Section
PORTA     EQU       $1000     Port A
PORTC     EQU       $1003     Port C
PORTD     EQU       $1008     Port D
DDRC      EQU       $1007     Data Direction for C
DDRD      EQU       $1009     Data Direction for D
BAUD      EQU       $102B     Baud Register (9600)
SCCR2     EQU       $102D     SCI Control Register 2
PACTL     EQU       $1026     For A7 direction
OPTION    EQU       $1039     Option Register
RTIVECT   EQU       $00EB     RTI Vector Location
TMSK2     EQU       $1024     RTI Enable Bit
TFLG2     EQU       $1025     RTI Flag Bit Register
OUTPUT    EQU       $FFAF     Send Byte Out
INPUT     EQU       $FFAC     Input Char or (0)
STACK     EQU       $01FF     Stack Pointer
BUFFALO   EQU       $E000     Buffalo's Starting Location
CRETN     EQU       $0D       Carriage Return
          ORG       $0000
```

```
* RAM Variables Section

FLAG:     RMB      1        Indicator for RTI System
IBUFF:    RMB      20       Input Buffer for Commands
COMPT:    RMB      1        Computer Designator (A-D)
CHANL:    RMB      1        Channel (1-4)
HILOW:    RMB      1        High Low for Writing (H, L)
DC_BYT:   RMB      1        Byte for D&C Output
BA_BYT:   RMB      1        Byte for A&B Output
DC_MSK:   RMB      1        Mask Byte for R/W on D&C
BA_MSK:   RMB      1        Mask Byte for R/W on B&A
U4:       RMB      1        Tri-State Data Read
U3:       RMB      1        Tri-State Data Read
LASTU4:   RMB      1        U4's Contents On Last Read
LASTU3:   RMB      1        U3's Contents On Last Read
XBIT:     RMB      1        Used With Don't Care States
          ORG      $B600

* This Subroutine does an Init to the System.
INIT:     LDS      #STACK   Set the Stack Pointer
          LDAA     #$0C     Set PD2 and PD3 to High
          STAA     PORTD    Store in PORTD
          LDAA     DDRD     Get Data Dir for D
          ORAA     #$0C     Make PD2 and PD3 Output
          STAA     DDRD     Store New Directions
          CLRA              A=$00
          STAA     PORTA    Strobes to Zero
          COMA              A=$FF
          STAA     PORTC    Port C to Zeros
          STAA     DDRC     Make All Pins on C Output
          LDAA     #$60     Strobe Mask for A
          STAA     PORTA    Make PA5 and PA6 High
          CLR      PORTA    Clear Strobes for Output
          CLR      DDRC     PORTC is now Input Only
          CLR      DC_BYT   Output Status Bytes
          CLR      BA_BYT
          JSR      READINP  Get Initial Values
          LDAA     U4       Make Last the Latest
          STAA     LASTU4
          LDAA     U3       Make Last the Latest
          STAA     LASTU3
          LDD      #RTISER  Address of RTI Service
          STD      $00EC    Rewrite Vector for RTI
          LDAA     PACTL    RTI Control Register
          ORAA     #$03     Make RTI 32mS
          STAA     PACTL    Store it Back
          LDAA     TMSK2    Set RTII to (1)
          ORAA     #$40     Mask Needed
          STAA     TMSK2    Store it Back
          LDAA     TFLG2    Get RTI RTIF Flag
          ORAA     #$40     Clear it
          STAA     TFLG2    Store it Back
          CLR      FLAG     RTI Permission Flag
          CLI               Allow Interrupts (RTI)
```

```
* Beginning of Program Loop
BEGNLP:   LDX     #IBUFF    Point X to Buffer Area
          CLR     FLAG      First Zero Flag
          INC     FLAG      RTI Permission to READINP
ILOOP:    JSR     SCINPUT   Go Get a Character
          STAA    0,X       Store the Character in Buffer
          INX               Point to Next Buffer Char
          CMPA    #$0D      Was Char a RETURN?
          BEQ     FNLOOP    If RETURN Char then Finish
          CPX     #IBUFF+20 Past End of Buffer?
          BNE     ILOOP     If Not Excessive Length Loop
          CLR     FLAG      End Permission READINP to RTI
          JMP     ERROR     Too Many Characters ERROR!
FNLOOP:   CLR     FLAG      End Permission for RTI READ
          LDX     #IBUFF    Repoint X to Start of Buffer
          LDAA    0,X       Load a Buffer Character
          INX               Point to Next Buff Character
          CMPA    #'R'      Is the Operation a READ?
          BNE     NOTRD     If Not a READ then Try WRITE
          JMP     READ      This is a READ Operation
NOTRD:    CMPA    #'W'      Was the Operation a WRITE?
          BNE     BADW      If a WRITE then Go There
          JMP     WRITE
BADW:     JMP     ERROR     It was Not a 'R' or 'W'


* RTI Service Routine.
RTISER:   LDAA    TFLG2     Get the Flags
          ORAA    #$40      Set to Clear RTIF Flag
          STAA    TFLG2     Clear Flag
          TST     FLAG      Test RTI Permission Flag
          BEQ     RTIEND    If FLAG=0 then RTI
          JSR     READINP   Read U4, U3
          LDAA    U3        Look at U3
          CMPA    LASTU3    Is U3 Same as Last Time?
          BEQ     SAMEU3    If Same Branch
          STAA    LASTU3    If Not Same then Make Equal
          BRA     NTSAME    Branch to Not Same Part
SAMEU3:     LDAA    U4        Look at U4
          CMPA    LASTU4    Is it the Same as Before?
          BEQ     RTIEND    If Same End RTI
          STAA    LASTU4    Store New U4 to Last U4
          BRA     NTSAME    Something Changed
RTIEND:   RTI               End RTI
NTSAME:   LDAA    #'I'      Beginning of Response
          JSR     OUTPUT    Send it
          LDAA    #'-'      Get the '-' for the Response
          JSR     OUTPUT    Send it
          LDAB    #16       There are 16 Lines to Read
RTILOOP:  PSHB              Save B
          LDD     U4        Get Both U4 and U3
          LSRD              Shift Right into Carry (CY)
          STD     U4        Store Back the Double Byte
          PULB              Restore B
          BCS     OUT_1     If CY=1 then Branch
          LDAA    #'0'      CY Must Have been a Zero
```

```
          JSR       OUTPUT    Send Loaded '0' to Host
          BRA       BTLP      Continue to RDAL1
OUT_1:    LDAA      #'1'      CY=1 So Send a '1' to Host
          JSR       OUTPUT    Send the '1'
BTLP:     DECB                Count down the 16 Bits
          BNE       RTILOOP   If Not Zero Yet then Do Again
          LDAA      #CRETN    Load a RETURN <CR>
          JSR       OUTPUT    Send CR
          RTI



***************************************************************
* * * * * * * * * * * * SUBROUTINES  * * * * * * * * * * * * *
***************************************************************

          ORG       $D000
* This program module is used to allow the Host Computer
*  to write to the Robots.  It is executed when a 'W'
*  Character is found as the first element in the IBUFF
WRITE:    LDAA      0,X       Get Next Character
          INX                 Point to Next Character
          CMPA      #'-'      Is the Character a '-'?
          BNE       WRT1      If Not a Write-All then Brch
          JMP       WRTALL    A Write-All Command
WRT1:     CMPA      #'A'      Check Lower Range
          BHS       WRT2      'A' or Higher is OK
          JMP       ERROR     Below 'A' is Not OK
WRT2:     CMPA      #'D'      Check for the 'D' Computer
          BLS       WRT3      'D' or Less Is OK
          JMP       ERROR     Above 'D' is Not OK
WRT3:     STAA      COMPT     Put 'A'-'D' in COMPT Location
          LDAA      0,X       Get Next Character
          INX                 Point to Next Buffer Element
WRT4:     CMPA      #'1'      Channel 1-4...  Compare to '1'
          BHS       WRT5      '1' or More is OK
          JMP       ERROR     ERROR if Below '1'
WRT5:     CMPA      #'4'      Compare to '4'
          BLS       WRT6      '4' or Less is OK
          JMP       ERROR     ERROR if Above '4'
WRT6:     STAA      CHANL     Record Channel Value for Later
          LDAA      0,X       Get Next Buffer Element
          INX                 Point to Next Buffer Element
          CMPA      #'L'      Is it a LOW?
          BEQ       WRGOOD    LOW is OK
          CMPA      #'H'      Is it a HIGH?
          BEQ       WRGOOD    HIGH is OK
          JMP       ERROR     Not HIGH or LOW!
WRGOOD:   STAA      HILOW     Store as 'H' or 'L'
          JSR       POSIT     Make DC_MSK and BA_MSK
          LDAA      HILOW     Get HILOW ('H' or 'L')
          CMPA      #'L'      Is HILOW = 'L'?
          BEQ       MKLOW     If So then Make Bit LOW
MKHIGH:   LDAA      DC_MSK    HILOW Must have Been 'H'
          ORAA      DC_BYT    Make Bit High
          STAA      DC_BYT    Store Back in DC_BYT
          LDAA      BA_MSK    Get B&A Mask Byte
```

```
         ORAA     BA_BYT    Make Bit High
         STAA     BA_BYT    Store Back in BA_BYT
         BRA      WRTIT     Go Write It
MKLOW:   LDAA     DC_MSK    Get DC Mask Byte
         COMA               Make 1's Compliment
         ANDA     DC_BYT    Make Bit Zero
         STAA     DC_BYT    Store Back
         LDAA     BA_MSK    Load Next Bit Mask
         COMA               1's Compliment
         ANDA     BA_BYT    Make Bit LOW
         STAA     BA_BYT    Store it Back
WRTIT:   COMA               Compliment Before Output
         STAA     PORTC     Put BA_BYT to PORTC
         BSR      STB_BA    Strobe PORTC to u1
         LDAA     DC_BYT    Load Next Byte
         COMA               Compliment Before Output
         STAA     PORTC     Store it to PORTC
         BSR      STB_DC    Strobe PORTC to u2
         JSR      WRT_BUF   Original Command to Host
         JMP      BEGNLP    Finished Echoing Input Back

* This routine strobes the information just previously written
*  to the C Port into U1 or U2
STB_BA:  SEI                Prohibits I-Type Interrupts
         LDAB     PORTD     Get Tri-State Strobe Byte
         ORAB     #$0C      Disable Both Strobes
         STAB     PORTD     Write It Back
         LDAB     #$FF      For DDRC
         STAB     DDRC      Make All PORTC Pins Output
         LDAB     PORTA     Get PORTA Strobes
         ORAB     #$40      Make PA6 HIGH
         STAB     PORTA     Store Strobes Back
         ANDB     #$BF      Make Bit PA6 LOW
         STAB     PORTA     Strobe Bit Now Zero Again
         CLR      DDRC      Sets PORTC to INPUT
         CLI                Lets Interrupts Occur Again
         RTS

* This routine strobes the information just previously written
*  to the C Port into U1 or U2
STB_DC:  SEI                Prohibits I-Type Interrupts
         LDAB     PORTD     Get Tri-State Strobe Byte

         ORAB     #$0C      Disable Both Strobes
         STAB     PORTD     Write It Back
         LDAB     #$FF      For DDRC
         STAB     DDRC      Make All PORTC Pins Output
         LDAB     PORTA     Get Strobes
         ORAB     #$20      Make PA5 High
         STAB     PORTA     Store it
         ANDB     #$DF      Make PA5 LOW
         STAB     PORTA     Strobe Bit Now Zero Again
         CLR      DDRC      Sets PORTC to INPUT
         CLI                Lets Interrupts Occur Again
         RTS

* This program module writes 16 0's or 1's to the entire system.
```

```
* Example:  To write to all 16 outputs enter the following
*  command or a variant:
*  W-1011000101111010<CR>  The first 4 bits are for Computer A
*                          The second 4 are for Computer B etc.
*                          A1=1, A2=0, A3=1, A4=1
WRTALL:   LDY      DC_BYT    Current Value of Output
          LDAB     #16       After '-' There are 16 0's or 1's
WRLOOP:   LDAA     0,X       Get The First 0 or 1
          INX                Point to Next Char in Buffer
          CMPA     #'X'      Don't Care Operator
          BEQ      BIT_OK    Don't Care is OK
          CMPA     #'0'      Is the Char a '0'?
          BEQ      BIT_OK    If So it is OK
          CMPA     #'1'      Is the Char a '1'?
          BEQ      BIT_OK    If So it is OK
          JMP      ERROR     A Char was NOT '0' or '1'
BIT_OK:   PSHA               Save A & B for Later
          PSHB
          XGDY               Exchange D and Y
          LSRD               Shift Output Info into CY
          BCS      BMB1      If CY=1 Set XBIT=$01
          CLR      XBIT      If Current Data=0 then XBIT=0
          BRA      XFOUND    Finished with XBIT Determine
BMB1:     CLR      XBIT      XBIT=1,  Start With XBIT=0
          INC      XBIT      XBIT=1
XFOUND:   XGDY               Current Data Back in Y
          PULB               Restore A & B
          PULA
          CMPA     #'X'      Was it a Don't Care?
          BNE      NOTANX    If NOT an 'X' Branch
          LDAA     XBIT      Was an 'X', Look at XBIT
NOTANX:   RORA               Right Most Bit of ASCII into (CY)
          ROR      DC_BYT    (CY) Into MSD of DC_BYT
          ROR      BA_BYT    LSB of DC_BYT Into MSD of BA_BYT
          DECB               One Char Processed.  Decr. (B)
          BNE      WRLOOP    Branch if (B) Not Zero. Get Another
          LDAA     0,X       Check for RETURN at End
          CMPA     #$0D      Is it a RETURN <CR>
          BEQ      WROK      If so GOOD!  Finish
          JMP      ERROR     Was Not a RETURN at End. Bad!
WROK:     LDAA     DC_BYT    Get Loaded Byte for D&C
          COMA               Compliment
          STAA     PORTC     Send it to Port C
          JSR      STB_DC    Strobe it Out.
          LDAA     BA_BYT    Get BA Byte
          COMA               Compliment
          STAA     PORTC     Send it to Port C
          JSR      STB_BA    Strobe it Out
          JSR      WRT_BUF   Success!  Write the Command to Host
          JMP      BEGNLP    Do it All Over Again.


* This subroutine writes out the contents of the input
*  buffer after a successful command execution.
WRT_BUF:  LDX      #IBUFF    Point (X) to Start of Buff
WREPLY:   LDAA     0,X       Get Byte from Buffer
          JSR      OUTPUT    Send it to Host Computer
```

```
         CMPA     #$0D      Was the Byte a RETURN?
         BEQ      WFIN      If Last Byte Sent a RETURN
         INX                Point to Next Buffer Element
         BRA      WREPLY    Do it Again
WFIN:    RTS

* This subroutine reads a char from INPUT in BUFFALO.  If there
*  is no char then INPUT returns a zero which causes this sub
*  to loop and look for a non-zero character.  The execution
*  stays here till a character is entered.
SCINPUT: JSR      INPUT     Buffalo Input
         TSTA               Was there an Input?
         BEQ      SCINPUT   If No Input Branch
         RTS                Finished

* This is the read module of the program.  This is both for
*  single reads and Read-Alls.
READ:    LDAA     0,X       Get Next Character
         INX                Point to Next Character
         CMPA     #'-'      Read All Channels
         BNE      RD1       If Not a '-' then Branch
         JMP      RDALL     Otherwise Read All
RD1:     CMPA     #'A'      Check Lower Range
         BHS      RD2       'A' or Higher is OK
         JMP      ERROR     Below 'A' is Not OK
RD2:     CMPA     #'D'      Check for the 'D' Computer
         BLS      RD3       'D' or Less is OK
         JMP      ERROR     Above 'D' is Not OK
RD3:     STAA     COMPT     Put 'A'-'D' in COMPT Location
         LDAA     0,X       Get Next Character
         INX                Point to Next Buffer Element
RD4:     CMPA     #'1'      Channel 1-4...  Compare to '1'
         BHS      RD5       '1' or More is OK
         JMP      ERROR     ERROR if Below '1'
RD5:     CMPA     #'4'      Compare to '4'
         BLS      RD6       '4' or Less is OK
         JMP      ERROR     ERROR if Above '4'
RD6:     STAA     CHANL     Record Channel Value for Later
         JSR      READINP   Load Contents into U3 and U4
         JSR      POSIT     Make DC_MSK and BA_MSK
         LDAA     DC_MSK    Get Mask Byte
         ANDA     U4        And with DC Memory Location
         STAA     U4        Store May or May not have a (1)
         LDAA     BA_MSK    Get BA Mask Byte
         ANDA     U3        AND it with U3's Data
         ADDA     U4        U3 or U4 (May) Have a (1)
         TSTA               Test A for Setting Flags
         BEQ      RZRO      Go Load a Zero
         LDAA     #'1'      Load the '1'
         BRA      RD7       Go Send a '1'
RZRO:    LDAA     #'0'      Load the '0'
RD7:     JSR      RETURN    Go Write it as a Response
         JMP      BEGNLP    Do the Whole Thing Over


* This program module is activated by an input of 'R-' followed
*  by a RETURN.  The successful response to the Host is:
```

```
*  R-00010110100101110101 where the left most (0) is A1 and the
*  right most (1) is D4
RDALL:    LDAA     #'R'     Get the 'R' for the Response
          JSR      OUTPUT   Send it
          LDAA     #'-'     Get the '-' for the Response
          JSR      OUTPUT   Send it
          JSR      READINP  Go Read Input into CD_BYT, BA_BYT
          LDAB     #16      There are 16 Lines to Read
RDALOOP:  PSHA              Store A and B
          PSHB
          LDD      U4       Get Both U4 and U3
          LSRD              Shift Right into Carry (CY)
          STD      U4       Store Back the Double Byte
          PULB              Restore A and B
          PULA
          BCS      OUT_ONE  If CY=1 then Branch
OUT_ZERO: LDAA     #'0'     CY Must Have been a Zero
          JSR      OUTPUT   Send Loaded '0' to Host
          BRA      RDAL1    Continue to RDAL1
OUT_ONE:  LDAA     #'1'     CY=1 So Send a '1' to Host
          JSR      OUTPUT   Send the '1'
RDAL1:    DECB              Count down the 16 Bits
          BNE      RDALOOP  If Not Zero Yet then Do Again
          LDAA     #CRETN   Load a RETURN <CR>
          JSR      OUTPUT   Send CR
          JMP      BEGNLP   Do the Big Loop Again

* This routine enables the strobes for U3 and U4 one at a time
*   and records the information in U3 and U4.  These locations
*   correspond to the ICs (Tri-State Devices) that capture the
*   information.
READINP:  CLR      DDRC     Makes C Input on ALL Pins
          LDAA     PORTD    Get D Port
          ANDA     #$FB     Make PD2 Low
          STAA     PORTD    Store it Back in Port D
          PSHA              Save Port D Bits For Later
          LDAA     PORTC    Capture Information for A&B
          COMA              Compliment for Controller
          STAA     U3       Store A&B in U3 Memory Location
          PULA              Get Port D Info Back in (A)
          ORAA     #$0C     PD2 and PD3 High (Disable U3, U4)
          STAA     PORTD    Store Back in Port D
          ANDA     #$F7     Make PD3 Low (Enable U4)
          STAA     PORTD    Store it Back
          LDAA     PORTC    Get U4 Data
          COMA              Compliment for Controller
          STAA     U4       Store it in U4
          LDAA     PORTD    Get PORTD with Strobes
          ORAA     #$0C     Make PD2 and PD3 High (Disable)
          STAA     PORTD    Store it
          RTS


* This subroutine takes COMPT and CHANL and fills memory
*  locations called DC_MSK and BA_MSK with a single (1) in
*  one of the 16 bit positions of this Double Variable.
*  This is used for both WRITE and READ Operations.
```

```
POSIT:    LDAA    COMPT     Computer (A-D)
          SUBA    #'A'      Computer 'A'=0,'B'=1,'C'=2,'D'=3
          LSLA              Multiply (A) By 4
          LSLA
          LDAB    CHANL     Load Channel Byte ('1' --> '4')
          SUBB    #'1'      Channel 1=0, 2=1, 3=2, 4=3
          ABA               (A) Now has Number of Mask Shifts
          TAB               (B) Now has # Mask Shifts
          LDY     #$0000    Zero (Y)
          ABY               (Y) has # of Mask Shifts
          CLRA              Zero (D)
          CLRB
          INCB              (D) = $0001
SHIFTS:   CPY     #0        Is (Y) = $0000?
          BEQ     EDSHF     If (Y)=0 then Finished here
          LSLD              If (Y)<>0 then Shift (D) Mask Left
          DEY               Decrement (Y)
          BRA     SHIFTS    Do Again Till (Y) = Zero
EDSHF:    STD     DC_MSK    Store (D) in DC_MSK and BA_MSK
          RTS

* Subroutine to send whatever is in (A) to the Host Computer
*  followed by a Carriage Return only.
RETURN:   JSR     OUTPUT    Send the Character
          LDAA    #$0D      Carriage RETURN
          JSR     OUTPUT    Send CR
          RTS

* This is a catch-all for all errors.  This is not a subroutine
*  but a common program segment.  After any error the program
*  vectors back to BEGNLP where another command is processed.
ERROR:    LDAA    #'E'      Error Indicator to be Returned
          JSR     RETURN    Send It
          JMP     BEGNLP    Do It Again
```