EKLEKTIK DESIGN

Micah Richards

Microcomputer Design Project

04/24/2018

Description of Project

The microcomputer design project is a task requiring the production of a single board computer from scratch. The project necessitates the selecting of IC chips, designing of a printed circuit board, programming of chip communication procedures, and the programming of a monitor program for user interactions. At a minimum, the computer shall have an RS-232 compatible data port (Tx Data, Rx Data, CTS, RTS, Signal and Chassis Ground) capable of operating at 19,200 baud, an 8-bit bi-directional parallel data port with “handshaking” capability (8 data lines, ground, one control line to and one control line from the external circuitry) (optional), 64K of RAM (random access memory), the capability of addressing 64K of memory space, and an “on-board” monitor program residing in EEPROM. The monitor program shall be capable of examining the contents of any memory location within the memory space, changing the contents of any random-access memory location within the memory space, examining and changing the contents of any of the registers within the MPU, transmitting and receiving data via all data ports, allowing a machine-coded program to be loaded into RAM from a data port, and executing a program that has been loaded into RAM. This report details a design meeting these specifications.

Hardware Description

General Introduction

The hardware is composed of 8 IC chips soldered to a custom printed circuit board manufactured by 4PCB. As errors were encountered, breadboards and perf-board based break out boards were used to provide space for necessary modifications.

Device and Chip Selections

This project makes use of the Motorola 68000 CPU for its primary functionality. For memory, Atmel AT28C256 chips were selected for the ROM, while ZMD U62256A chips were selected for the RAM. Chip communication is made possible using a Xilinx XC9572 CPLD, and serial communications is enabled via a Philips SCC68681 DUART. The choice of these ICs was based on peer recommendations and past student's recommendations for the given task.

Hardware Block Diagram

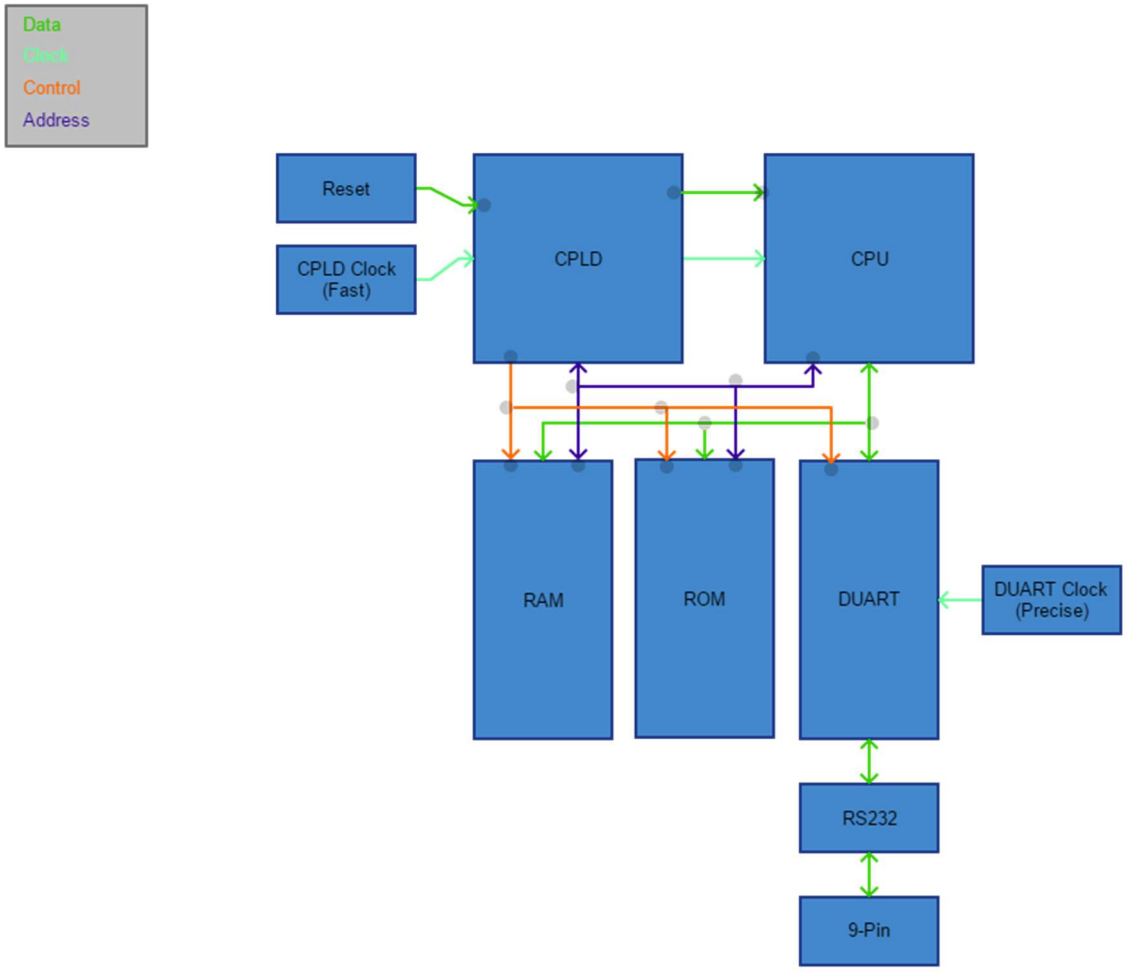
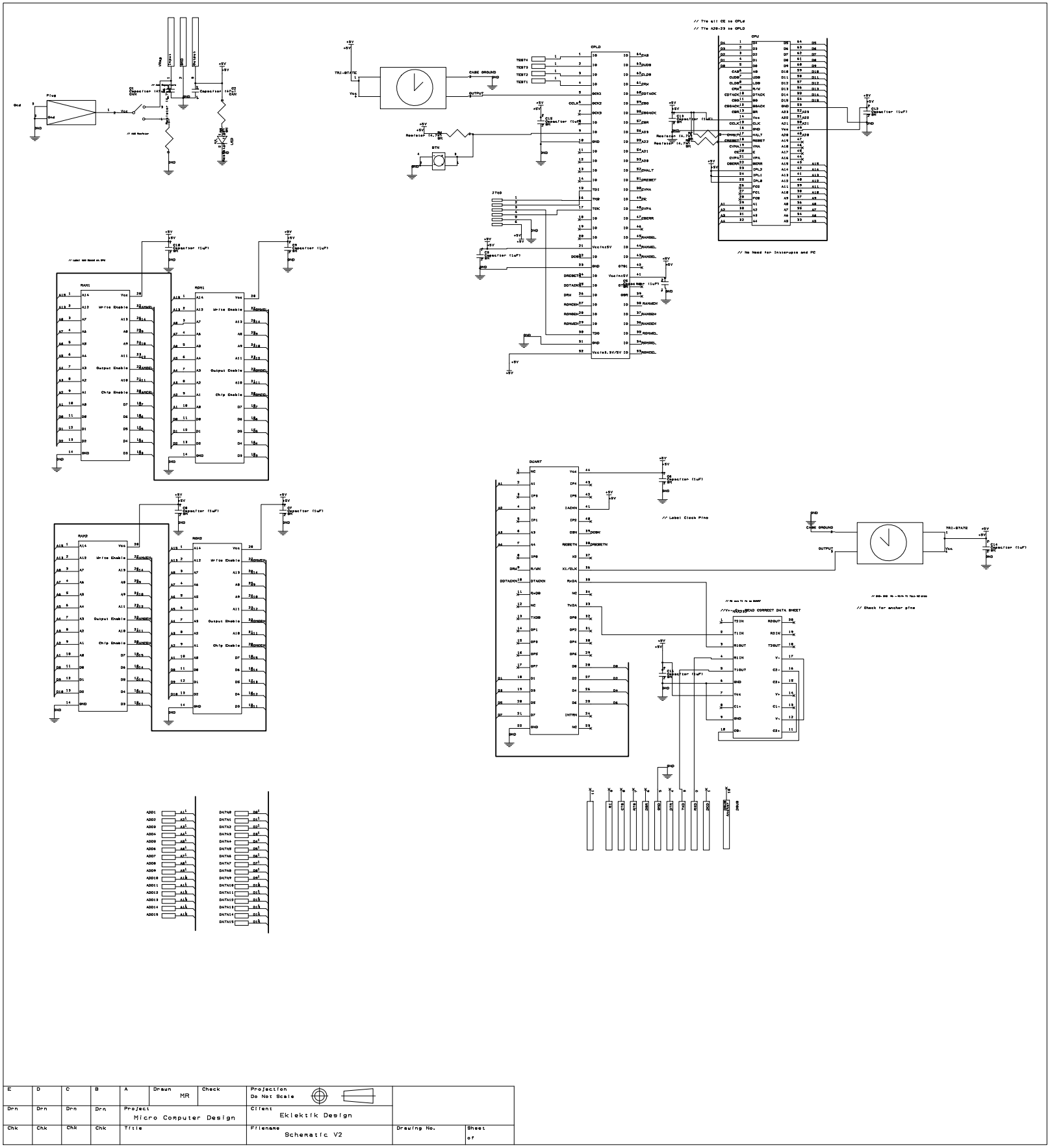
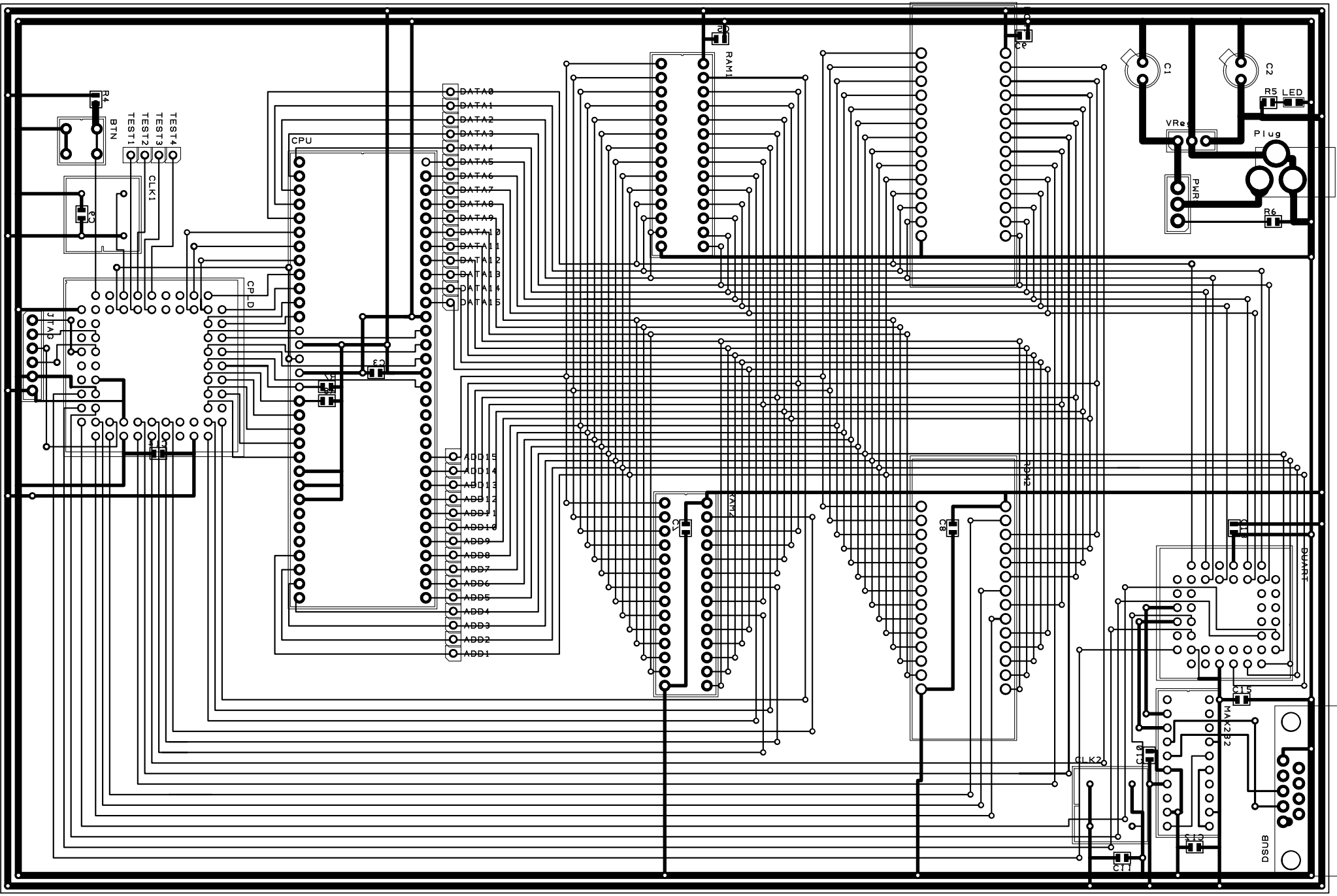


Figure 1 The initial conception of the linking between ICs

PCB Schematic



PCB Printout



Price Breakdown By Component

Digikey			E-Bay			Amazon			4PCB		
Quantity	Part	Price	Quantity	Part	Price	Quantity	Part	Price	Quantity	Part	Price
2	Switch	$ 1.16	1	45 Piece DIP Socket set	$ 14.57	120	Breadboard Jumper Wires	$ 6.99	1	PCB	$ 33.00
2	Power Jack	$ 1.20	2	ZIF Socket	$ 14.57					Shipping	$ 22.09
2	3.6864Mhz DUART Clock	$ 4.94									
2	80MHz CPLD Clock	$ 4.74									
2	D-SUB Plug	$ 3.40									
1	Max 233 IC	$ 5.68									
2	SRAM IC	$ 5.76									
2	EEPROM	$ 13.52									
2	68 Pin PLCC Socket	$ 2.88									
2	44 Pin PLCC Socket	$ 2.06									
25	1uF Capcitor	$ 1.98									
25	4.7kΩ Resistor	$ 7.55									
2	84 Pin PLCC Socket	$ 2.76									
3	Strip of 40 Header Pins	$ 1.53									
25	Green LED	$ 3.89									
10	Red LED	$ 3.96									
10	Red LED	$ 3.96									
20	150Ω Resistor	$ 6.04									
	Shipping	$ 10.65									
		$ 87.66									
	Tax	$ 7.23									
Totals		$ 94.89			$ 29.14			$ 6.99			$ 55.09

Final Total	$ 186.11

Software Description

General Introduction

The software for this project is divided into two sub-components. The first is the chip selector logic, written in VHDL, allowing the CPU to communicate with the RAM, ROM, and DUART. The second portion is the monitor code, written in 68000 Assembly, allowing a user to interface with the micro-computer via another computer's serial port.

Memory Mapping

The chip selection portion of the software uses a simple flow of conditional statements to enable the proper chip at the proper time. Each cycle, every chip is deactivated; a comparison is then made of the CPU's current outputs, and the proper chip is enabled. For this device, address lines 20-23 are tied to the CPLD for the purpose of selecting the device targeted by the CPU. In this design, 'F' signifies a selection of the RAM chips, '0' signifies the selection of the ROM chips, and '3' a selection of the DUART. Further differentiation on which chip needs selection is provided by testing the CPU's Address High and Address Low pins.

Detailed Description of Core Code

The monitor program is composed of seven unique features which can be selected by the user. At its beginning, the software resets all registers used in its operation. This is primarily to avoid mistakes in leftover data interfering with later operations. Following this, the menu is printed and the user is prompted for a selection.

Read From ROM

The first menu option allows the reader to read 4 bytes from a specified memory address in ROM. The user is prompted for an address and the program displays the contents at the given memory address.

Read From RAM

The second menu option functions in much the same way as the first, with the exception that the program appends a '0F' to the address rather that '00' to retrieve contents from RAM.

Read From A Register

The third section of the program has a different operation than the first two, as it prompts the user for a register rather than a location in memory. This is done by copying a selected registers contents into the D0 register, printing the contents, and then restoring the CPU to its previous state.

Write To RAM

The fourth option is the first to allow data to be written, enabling a user to modify data in the RAM. Like Read From RAM, the program prompts the user for an address and prepends the necessary '0F' to the address. Following this, the program prompts the user for 4 bytes via 8 ascii characters '0'-'F' which are then written to the given location.

Write To A Register

The fifth option allows the user to modify a register's contents. Note, some registers will not appear to change due to their reset at the beginning of the menu printout. Additionally, in some situations, more than just the selected register will be modified to ensure the stack is restored properly at the end of the

procedure. The program will prompt the user for a register from ‘A0’-‘A7’ or ‘D0’-‘D7’, then request 8 ascii characters for the new memory contents.

Write S-Record

The sixths option allows the user to load an S-Record into memory. This device accepts only S2 and S8 records, with any other type causing an error and returning the user to the menu. Invalid data will also cause an error. The program will prompt for the data to be entered, and will exit after the S8 record concludes the upload.

Run S-Record

The seventh option jumps the program to the beginning of the S-Record program. The uploaded program will be executed, and the user will then be returned to the main menu.

Flow Chart

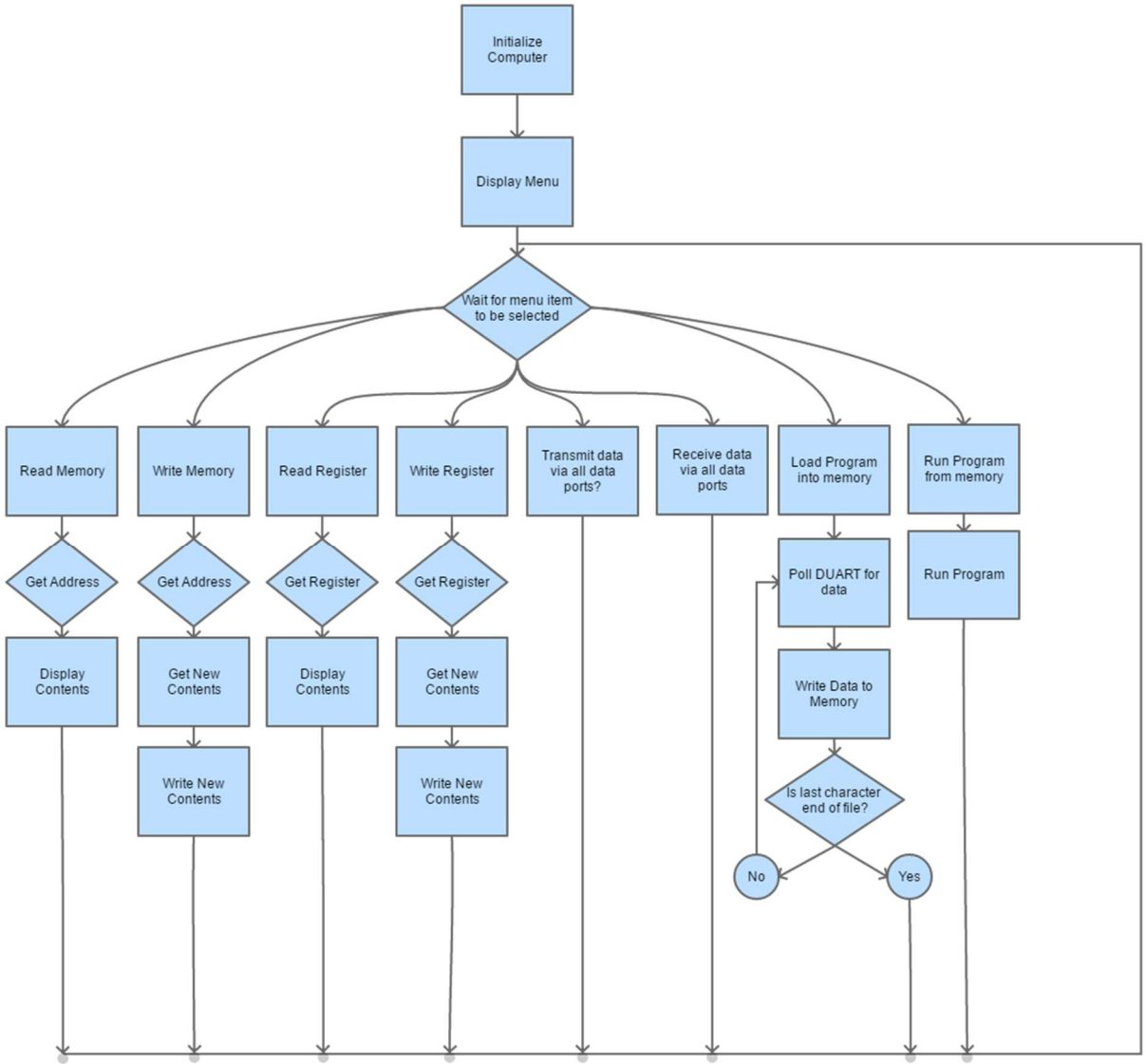


Figure 2 The Initial Concept For The Flow Of The Monitor Program

```
*------------------ Monitor Code ------------------------------------
* Title      : Monitor Code
* Written by : Micah Richards
*              Eklektik Design
* Date       : 04/02/18
* Description: Allows one to read and write from memory as well as
*               execute an S-Record
*--------------------------------------------------------------------

*------------------ Register Usages ---------------------------------
* D0 - Passed input to function
* D1 - Output from function
* D2 - Counter
* A0 - DUART Communications
* A2 - Memory writing target address
* A4 - String pointer temporary
* A7 - Stack pointer

*------------------ Constants ---------------------------------------
*---- Characters ----------
_CR         EQU     $0D                     * Carriage Return
_LF         EQU     $0A                     * Line Feed

*---- DUART ---------------
* (Note the offsets to account for no A0)
_DUART      EQU     $00300000               * Loaded in A0 when needed, regs are offsets
_MR1A       EQU     1                       * Mode Register1
_MR2A       EQU     1                       * Points here after MR1A is set
_SRA        EQU     3                       * Status Register (read)
 CSRA       EQU     3                       * Clock Select Register
_CRA        EQU     5                       * Command Register
_TBA        EQU     7                       * Transfer Holding Register
_RBA        EQU     7                       * Receive Holding Register
_ACR        EQU     9                       * Auxiliary control register
_RxRDY      EQU     0                       * Recieve ready bit position
_TxRDY      EQU     2                       * Transmit ready bit position
_BAUD       EQU     $CC                     * Baud rate value = 19,200 baud

*---- Memory Locations ----
_LRAM       EQU     $00F0FFFF               * RAM origin
_LROM       EQU     $0000FFFF               * ROM origin
_MORG       EQU     $00001000               * Monitor origin
_STACK      EQU     $00F00800               * Stack Pointer

*---- Strings -------------
            ORG     $00000100
_AREQ       DC.B    'Please enter a valid 2 byte address: ',                    0,  0
_AACC       DC.B    'Address Accepted!',                                        0,  0
_CONT       DC.B    'The memory contents at your selected index are: $',        0,  0
_CRLF       DC.B    _CR,_LF,                                                    0,  0
_ERRI       DC.B    'The input you submitted is invalid in this context.',      0,  0
_MENU       DC.B    'Welcome to the Eklektik Design uComputer Monitor Program', 0,  0
_MEN2       DC.B    'Select an option below:',                                  0,  0
_NDAT       DC.B    'Please enter the new data for this address: $',            0,  0
 OPT1       DC.B    '1) READ  from ROM',                                        0,  0
_OPT2       DC.B    '2) READ  from RAM',                                        0,  0
_OPT3       DC.B    '3) READ  from Register',                                   0,  0
_OPT4       DC.B    '4) WRITE to   RAM',                                        0,  0
_OPT5       DC.B    '5) WRITE to   Register',                                   0,  0
_OPT6       DC.B    '6) WRITE to   S-Record',                                   0,  0
 OPT7       DC.B    '7) RUN   the  S-Record',                                   0,  0
_PRMT       DC.B    'Eklektik@uComp:~$ ',                                       0,  0
_RREC       DC.B    'Please enter the S-record: ',                              0,  0
_RREQ       DC.B    'Please select a register (A0-A7, D0-D7): ',                0,  0

*------------------ Functions ---------------------------------------

*------------------ Main Code ---------------------------------------
* Purpose: Initialize the monitor program and provide an entry point
*          for resets
* In:  NA
* Out: NA
*--------------------------------------------------------------------
            ORG     $00000000               * Hard Reset
            DC.L    _STACK                  * Initialize the stack
            DC.L    BEGIN

            ORG     _MORG

BEGIN       JSR     INIT_DUART

RESET       MOVE.L  #$00000000, D0          * Reset D0
            MOVE.L  #$00000000, D1          * Reset D1
            MOVE.L  #$00000000, A4          * Reset A4
            MOVE.L  #$00F00800, A7          * Reset A7

            JSR     PRT_MENU                * Print Menu
            JSR     GET_CHAR                * Get input
            JSR     RUN_OPT                 * Run user's selection
            JMP     RESET                   * Repeat indefinitely


*
*------------------ ASC_ADDR ----------------------------------------
* Purpose: converts the lower two bytes of a register to ascii
* In:  D0
* Out: D1, PRT_****
* Note: A4 overwritten
*--------------------------------------------------------------------
ASC_ADDR    MOVE.L  D2,         -(A7)       * Store working register
            MOVE.L  D0,         -(A7)       * Store working register

            MOVE.B  #$04,       D2          * Initialize the counter

ASC_ADDR_L
            LSL.L   #8,         D1          * Shift to make space for next input
            BSR     MAKE_TEX                * Try to convert the input to hex
            LSR.W   #4,         D0          * Shift to make space
            SUBI.B  #$01,       D2          * Decrement
            BNE     ASC_ADDR_L              * Restart loop if needed
            MOVE.L  (A7)+,      D0          * Restore working register
            MOVE.L  (A7)+,      D2          * Restore working register
```

```
            RTS                             * Return

*------------------ GET_ADDR ---------------------------------------
* Purpose: Gets an address from the user
* In:  D0
* Out: D1, PRT_****
* Note: A4 overwritten
*-----------------------------------------------------------------
GET_ADDR    MOVE.L  D0,         -(A7)       * Store working register
            LEA     _AREQ,      A4          * Load address prompt
            BSR     PRT_NWLN                * Print a new line
            BSR     PRT_STRG                * Print string
            BSR     GET_FOUR                * Get the address to read
            MOVE.W  #$0001,     D0          * Move 1 into D1
            AND.B   D1,         D0          * Isolate last bit of D0
            BNE     PRT_ERR                 * If address is odd, Err

            BSR     PRT NWLN                * Print new line
            LEA     _AACC,      A4          * Else load address accept message
            BSR     PRT_STRG                * Display message
            BSR     PRT_NWLN                * Print new line
            BSR     PRT_NWLN                * Print new line
            MOVE.L  (A7)+,      D0          * Restore working register
            RTS                             * Return

*------------------ GET_CHAR ---------------------------------------
* Purpose: Gets a character through the serial connection and places
*          it in D0
* In:  IO
* Out: D0
*-----------------------------------------------------------------
GET_CHAR    LEA     _DUART,     A0          * A0 points to base DUART address
            MOVE.B  _SRA(A0),   D0          * Read the A status register
            BTST    #_RxRDY,    D0          * Test reciever ready status
            BEQ     GET_CHAR                * UNTIL char recieved
            MOVE.B  _RBA(A0),   D0          * Read the character into D0
            RTS                             * Return

*GET_CHAR    MOVE.L  D1,         -(A7)       * Save working register
*            MOVE    #5,         D0          * Store 'get single char' command
*            TRAP    #15                     * Trigger simulator action
*            MOVE    D1,         D0          * Move result into output register
*            MOVE.L  (A7)+,      D1          * Restore working register
*            RTS                             * Return

*------------------ GET_FOUR ---------------------------------------
* Purpose: Gets a four ascii values and saves them as hex
* In:  GET_CHAR
* Out: D1
*-----------------------------------------------------------------
GET_FOUR    MOVE.L  D2,         -(A7)       * Store working register
            MOVE.L  D0,         -(A7)       * Store working register
            MOVE.B  #$04, D2                * Initialize the counter

GET FOUR L  LSL.W   #4,         D1          * Shift to make space for next input
            BSR     GET_CHAR                * Get the next char
            BSR     MAKE_HEX                * Try to convert the input to hex
            SUBI.B  #$01,       D2          * Decrement
            BNE     GET_FOUR_L              * Restart loop if needed

            MOVE.L  (A7)+,      D0          * Restore working register
            MOVE.L  (A7)+,      D2          * Restore working register

            RTS                             * Else return

*------------------ INIT_DUART -------------------------------------
* Purpose: Initialize the DUART
* In:  N/A
* Out: N/A
*-----------------------------------------------------------------

INIT_DUART  LEA     _DUART,     A0          * A0 points to base DUART address

*---- Software Reset ------
            MOVE.B  #$30,       _CRA(A0)    * Reset TxA
            MOVE.B  #$20,       _CRA(A0)    * Reset RxA
            MOVE.B  #$10,       _CRA(A0)    * Reset MRA pointer

*---- Initialization ------
            MOVE.B  #$80,       _ACR(A0)    * selects baud rate set 2
            MOVE.B  #_BAUD,     _CSRA(A0)   * set 19.2k baud Rx/Tx
            MOVE.B  #$13,       _MR1A(A0)   * 8-bits, no parity, 1 stop bit

* This is the most important register to set in the 68681 DUART.
* 07 sets: Normal mode, CTS and RTS disabled, stop bit length = 1
* For testing load $#47 to enable auto-echo
            MOVE.B  #$07,       _MR2A(A0)

            MOVE.B  #$05,       _CRA(A0)    * enable Tx and Rx
            RTS

*------------------ MAKE HEX ---------------------------------------
* Purpose: Converts a value to hex if is 0-9 or A-F
* In:  D0
* Out: D1, C Flag
*-----------------------------------------------------------------
*---- 0-9 ------
MAKE HEX    CMP.B   #$30,       D0          * Compare input with $30
            BLT     MAKE_HEX_E              * Set error flag if less
            CMP.B   #$39,       D0          * Compare input with $40
            BGT     MAKE_HEX_2              * Check if A-F
            SUBI.B  #$30,       D0          * Subtract $30 from input
            JMP     MAKE_HEX_X              * Jump to exit

*---- A-F ------
MAKE_HEX_2  CMP.B   #$41,       D0          * Compare input with $41
            BLT     MAKE_HEX_E              * Set error flag if less
            CMP.B   #$46,       D0          * Compare input with $46
            BGT     MAKE_HEX_3              * Check if a-f
            SUBI.B  #$37,       D0          * Subtract $37 from input
            JMP     MAKE_HEX_X              * Jump to exit
```

```
*---- a-f ------
MAKE_HEX_3  CMP.B   #$61,       D0          * Compare input with $61
            BLT     MAKE_HEX_E              * Set error flag if less
            CMP.B   #$66,       D0          * Compare input with $66
            BGT     MAKE_HEX_E              * Set error flag if greater
            SUBI.B  #$57,       D0          * Subtract $57 from input
            JMP     MAKE_HEX_X              * Jump to exit

*---- Error ------
MAKE_HEX_E  JMP     PRT_ERR                 * Error and reset


*---- Exit ------
MAKE_HEX_X  OR.B    D0,         D1          * Move results to D1
            RTS                             * Return

*------------------ MAKE TEX ---------------------------------------
* Purpose: Converts a hex value to text
* In:  D0
* Out: D1, C Flag
*-----------------------------------------------------------------
*---- 0-9 ------
MAKE TEX    MOVE.B  #$0F,       D1
            AND.B   D0,         D1
            CMP.B   #$09,       D1          * Compare input with $09
            BGT     MAKE_TEX_2              * Check if A-F
            ADDI.B  #$30,       D1          * Add $30 to input
            RTS                             * Return

*---- A-F ------
MAKE_TEX_2  CMP.B   #$0F,       D1          * Compare input with $46
            BGT     MAKE_TEX_E              * Check if a-f
            ADDI.B  #$37,       D1          * Add $37 from input
            RTS                             * Return

*---- Error ------
MAKE_TEX_E  JMP     PRT_ERR                 * Error and reset

*------------------ PRT_2BYT ---------------------------------------
* Purpose: Prints the lower two bytes of a register
* In:  D1
* Out: PRT_****
* Note: A4 overwritten
*-----------------------------------------------------------------
PRT_2BYT    MOVE.L  D0,         -(A7)       * Save working register
            MOVE.L  D1,         -(A7)       * Save working register
            *MOVE.L D1,         D0          * Move ouput to input
            BSR     ASC_ADDR                * Convert to ascii
            MOVE.L  D1,         D0          * Move ouput to input

            BSR     PUT_CHAR                * Print first char
            LSR.L   #8,         D0          * Shift for next char

            BSR     PUT CHAR                * Print second char
            LSR.L   #8,         D0          * Shift for next char
            BSR     PUT_CHAR                * Print third char
            LSR.L   #8,         D0          * Shift for next char

            BSR     PUT_CHAR                * Print fourth char

            MOVE.L  (A7)+,      D1          * Restore D1
            MOVE.L  (A7)+,      D0          * Restore D0

            RTS                             * Return

*------------------ PRT_ERR ----------------------------------------
* Purpose: Prints the error message
* In:  N/A
* Out: PRT_****
* Note: A4 overwritten
*-----------------------------------------------------------------
PRT_ERR     BSR     PRT_NWLN                * Print a new line
            LEA     _ERRI,      A4          * Else load error message
            BSR     PRT_STRG                * Print it
            BSR     PRT_NWLN                * Print a new line
            BSR     PRT_NWLN                * Print a new line
            BRA     RESET                   * Return to menu

*------------------ PRT_MENU ---------------------------------------
* Purpose: Prints the monitor menu
* In:  N/A
* Out: PRT_****
* Note: A4 overwritten
*-----------------------------------------------------------------
PRT_MENU    MOVEM.L A4,         -(A7)       * Save working register

            BSR     PRT_NWLN                * Print new line

            LEA     _MENU,      A4          * Point to first menu string
            BSR     PRT STRG                * Print it
            BSR     PRT_NWLN                * Print new line

            LEA     _MEN2,      A4          * Point to second menu string
            BSR.S   PRT_STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT1,      A4          * Point to first option string
            BSR.S   PRT_STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT2,      A4          * Point to second option string
            BSR.S   PRT STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT3,      A4          * Point to third option string
            BSR.S   PRT_STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT4,      A4          * Point to fourth option string
            BSR.S   PRT_STRG                * Print it
```

```
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT5,      A4          * Point to fifth option string
            BSR.S   PRT_STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT6,      A4          * Point to sixth option string
            BSR.S   PRT_STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line

            LEA     _OPT7,      A4          * Point to seventh option string
            BSR.S   PRT_STRG                * Print it
            BSR.S   PRT_NWLN                * Print new line


            LEA     _PRMT,      A4          * Point to prompt string
            BSR.S   PRT_STRG                * Print it

            MOVEM.L (A7)+,      A4          * Restore working register
            RTS                             * Return

*------------------ PRT_NWLN ---------------------------------------
* Purpose: Prints a new line through the serial connection
* In:  N/A
* Out: PRT_STRG
*-------------------------------------------------------------------
PRT_NWLN    MOVEM.L A4,         -(A7)       * Save working register
            LEA     _CRLF,      A4          * Point to CR/LF string
            BSR.S   PRT_STRG                * Print it
            MOVEM.L (A7)+,      A4          * Restore working register
            RTS                             * Return

*------------------ PRT_REG ----------------------------------------
* Purpose: Prints the contents of a register in ascii
* In:  D1
* Out: PRT_****
*-------------------------------------------------------------------
PRT_REG     MOVE.L  D1,         -(A7)       * Save working register
            LEA     _CONT,      A4          * Load Memory contents message
            BSR     PRT_STRG                * Print message
            SWAP    D0                      * Swap high and low
            BSR     PRT_2BYT                * Print the data
            LSR.L   #8,         D0          * Shift upper bits down
            LSR.L   #8,         D0          * Shift upper bits down
            BSR     PRT_2BYT                * Print the data
            BSR     PRT_NWLN                * Print new line
            BSR     PRT_NWLN                * Print new line
            MOVE.L  (A7)+,      D1          * Restore D1
            RTS                             * Return

*------------------ PRT_STRG ---------------------------------------
* Purpose: Prints a string through the serial connection
* In:  A4
* Out: PUT_CHAR
* Note: A4 Destroyed
*-------------------------------------------------------------------
PRT_STRG    MOVE.L  D0,         -(A7)       * Save working register
PRT_STRG_1  MOVE.B  (A4)+,      D0          * Get character to be printed
            BEQ.S   PRT_STRG_2              * If null then return
            BSR     PUT_CHAR                * Else print it
            BRA     PRT STRG 1              * Continue
PRT_STRG_2  MOVE.L  (A7)+,      D0          * Restore D0
            RTS                             * Return

*------------------ PUT_CHAR ---------------------------------------
* Purpose: Sends a character through the serial connection
* In:  D0
* Out: IO
*-------------------------------------------------------------------
PUT_CHAR    LEA     _DUART,     A0          * A0 points to base DUART address
            MOVE.W  D0,         -(SP)
PUT_CHAR_L  MOVE.B  _SRA(A0),   D0
            BTST    #_TxRDY,    D0
            BEQ     PUT_CHAR_L
            MOVE.W  (SP)+,      D0
            MOVE.B  D0,         _TBA(A0)
            RTS                             * Return

*PUT_CHAR    MOVE.L  D0,         -(A7)       * Save working regiser
*            MOVE.L  D1,         -(A7)       * Save working register
*            MOVE.L  D0,          D1         * Move information to D1
*            MOVE    #6,          D0         * Load Printchar trap routine
*            TRAP    #15                     * Call simulator procedure
*            MOVE.L  (A7)+,       D1         * Restore working register
*            MOVE.L  (A7)+,       D0         * Restore working register
*            RTS                             * Return

*------------------ READ_RAM ---------------------------------------
* Purpose: Reads data from a given RAM address
* In:  GET_ADDR
* Out: PRT_****
*-------------------------------------------------------------------
READ_RAM    MOVE.L  D0,         -(A7)       * Save working register
            MOVE.L  A2,         -(A7)       * Save working register

            BSR     GET_ADDR                * Get the target address
            MOVE.L  #_LRAM,     D0          * Load first two bytes with RAM target
            AND.W   D1,         D0          * Set the last two bytes to specific address
            MOVE.L  D0,         A2          * Move the data to the address register
            MOVE.L  (A2),       D0          * Move the data from memory to D0

            BSR     PRT_REG                 * Print the contents
            MOVE.L  (A7)+,      A2          * Restore working register
            MOVE.L  (A7)+,      D0          * Restore working register
            RTS                             * Return

*------------------ READ_REG ---------------------------------------
* Purpose: Reads data from a given register
* In:  GET_CHAR
* Out: PRT_****
*-------------------------------------------------------------------
READ_REG    MOVE.L  D0,         -(A7)       * Save working register
```

```
            MOVE.L  A4,         -(A7)       * Save working register
            BSR     PRT_NWLN                * Print new line
            LEA     _RREQ,      A4          * Load register prompt
            BSR     PRT_STRG                * Print message
            MOVE.L  (A7)+,      A4          * Restore working register

            BSR     GET_CHAR                * Get the address to read
            CMP     #$41,       D0          * Check if 'A'
            BEQ     READ_REG_A
            CMP     #$61,       D0          * Check if 'a'
            BEQ     READ_REG_A

            CMP     #$44,       D0          * Check if 'D'
            BEQ     READ_REG_D
            CMP     #$64,       D0          * Check if 'd'
            BEQ     READ_REG_D
            JMP     PRT_ERR

READ_REG_A  BSR     GET_CHAR                * Get the address to read
READ_REG_A0 CMP.B   #$30,       D0          * Check if '0'
            BNE     READ_REG_A1             * Jump to next test
            MOVE.L  A0,         D0          * Else move A0 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A1 CMP.B   #$31,       D0          * Check if '1'
            BNE     READ_REG_A2             * Jump to next test
            MOVE.L  A1,         D0          * Else move A1 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A2 CMP.B   #$32,       D0          * Check if '0'
            BNE     READ_REG_A3             * Jump to next test
            MOVE.L  A2,         D0          * Else move A2 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A3 CMP.B   #$33,       D0          * Check if '0'
            BNE     READ_REG_A4             * Jump to next test
            MOVE.L  A3,         D0          * Else move A3 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A4 CMP.B   #$34,       D0          * Check if '0'
            BNE     READ_REG_A5             * Jump to next test
            MOVE.L  A4,         D0          * Else move A4 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A5 CMP.B   #$35,       D0          * Check if '0'
            BNE     READ_REG_A6             * Jump to next test
            MOVE.L  A5,         D0          * Else move A5 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A6 CMP.B   #$36,       D0          * Check if '0'
            BNE     READ_REG_A7             * Jump to next test
            MOVE.L  A6,         D0          * Else move A6 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_A7 CMP.B   #$37,       D0          * Check if '0'
            BNE     READ_REG_E              * Jump to next test
            MOVE.L  A7,         D0          * Else move A7 to D0
            JMP     READ_REG_P              * Jump to printing


READ_REG_D  BSR     GET_CHAR                * Get the address to read
READ_REG_D0 CMP.B   #$30,       D0          * Check if '0'
            BNE     READ_REG_D1             * Jump to next test
            MOVE.L  (A7)+,      D0          * Restore working register
            JMP     READ_REG_P              * Jump to printing
            BSR     PRT NWLN                * Print new line
            BSR     PRT_REG                 * Print the register
            RTS                             * Return

READ_REG_D1 CMP.B   #$31,       D0          * Check if '1'
            BNE     READ_REG_D2             * Jump to next test
            MOVE.L  D1,         D0          * Else move D1 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_D2 CMP.B   #$32,       D0          * Check if '0'
            BNE     READ_REG_D3             * Jump to next test
            MOVE.L  D2,         D0          * Else move D2 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_D3 CMP.B   #$33,       D0          * Check if '0'
            BNE     READ_REG_D4             * Jump to next test
            MOVE.L  D3,         D0          * Else move D3 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_D4 CMP.B   #$34,       D0          * Check if '0'
            BNE     READ_REG_D5             * Jump to next test
            MOVE.L  D4,         D0          * Else move D4 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_D5 CMP.B   #$35,       D0          * Check if '0'
            BNE     READ REG D6             * Jump to next test
            MOVE.L  D5,         D0          * Else move D5 to D0
            JMP     READ_REG_P              * Jump to printing

READ_REG_D6 CMP.B   #$36,       D0          * Check if '0'
            BNE     READ_REG_D7             * Jump to next test
            MOVE.L  D6,         D0          * Else move D6 to D0
            JMP     READ_REG_P               * Jump to printing

READ_REG_D7 CMP.B   #$37,       D0          * Check if '0'
            BNE     READ_REG_E              * Jump to next test
            MOVE.L  D7,         D0          * Else move D7 to D0

READ_REG_E  JMP     PRT_ERR                 * Jump to printing

READ_REG_P  BSR     PRT_NWLN                * Print new line
            BSR     PRT_REG                 * Print the register
            MOVE.L  (A7)+,      D0          * Restore working register

            RTS                             * Return
```

```
*------------------ READ_ROM ---------------------------------------
* Purpose: Reads data from a given ROM address
* In:  D1
* Out: PRT_****
* Note: A4 overwritten
*-------------------------------------------------------------------

READ_ROM    MOVE.L  D0,         -(A7)       * Save working register
            MOVE.L  A2,         -(A7)       * Save working register


            BSR     GET_ADDR                * Get target address

            MOVE.L  #_LROM,     D0          * Load first two bytes with ROM target
            AND.W   D1,         D0          * Set the last two bytes to specific address
            MOVE.L  D0,         A2          * Move the data to the address register
            MOVE.L  (A2),       D0          * Move the data from memory to D0

            BSR     PRT_REG                 * Print the contents
            MOVE.L  (A7)+,      A2          * Restore working register
            MOVE.L  (A7)+,      D0          * Restore working register
            RTS                             * Return

*------------------ RITE RAM ---------------------------------------
* Purpose: Writes data to a given RAM address
* In:  GET_ADDR, GET_FOUR
* Out: Data to RAM address
*-------------------------------------------------------------------
RITE_RAM    MOVE.L  D0,         -(A7)       * Save working register
            MOVE.L  A2,         -(A7)       * Save working register

            BSR     GET_ADDR                * Get target address
            MOVE.L  #_LRAM,     D0          * Load first two bytes with RAM target
            AND.W   D1,         D0          * Set the last two bytes to specific address
            MOVE.L  D0,         A2          * Move the data to the address register

            LEA     _NDAT,      A4          * Load New Ram Prompt
            BSR     PRT_STRG

            BSR     GET_FOUR                * Get the first four data points
            LSL.L   #8,         D1          * Shift the data over to make space
            LSL.L   #8,         D1          * Shift the data over to make space
            BSR     GET_FOUR                * Get the latter four data points
            MOVE.L  D1,         (A2)        * Move the data to memory
            BSR     PRT_NWLN                * Print new line
            BSR     PRT_NWLN                * Print new line

            MOVE.L  (A7)+,      A2          * Restore working register
            MOVE.L  (A7)+,      D0          * Restore working register
            RTS                             * Return

*------------------ RITE_REG ---------------------------------------
* Purpose: Writes data to a selected register
* In:  GET_CHAR, GET_FOUR
* Out: Data to selected register
*-------------------------------------------------------------------
RITE_REG    MOVE.L  D0,         -(A7)       * Save working register
            MOVE.L  D1,         -(A7)       * Save working register

            BSR     PRT_NWLN                * Print new line
            LEA      RREQ,      A4          * Load register prompt
            BSR     PRT_STRG                * Print message
            BSR     GET_CHAR                * Get the register type to read
            CMP.B   #$41,       D0          * Check if 'A'
            BEQ     RITE_REG_A
            CMP.B   #$61,       D0          * Check if 'a'
            BEQ     RITE_REG_A

            CMP.B   #$44,       D0          * Check if 'D'
            BEQ     RITE_REG_D
            CMP.B   #$64,       D0          * Check if 'd'
            BEQ     RITE_REG_D
            JMP     PRT_ERR                 * Else error and return

RITE_REG_A  BSR     GET_CHAR                * Get the register # to read
            BSR     PRT_NWLN                * Print a new line
            LEA     _NDAT,      A4          * Load new data Prompt
            BSR     PRT_STRG                * Print the message

            BSR     GET_FOUR                * Get the first four data points
            LSL.L   #8,         D1          * Shift the data over to make space
            LSL.L   #8,         D1          * Shift the data over to make space
            BSR     GET_FOUR                * Get the latter four data points
            BSR     PRT_NWLN                * Print a new line
            BSR     PRT_NWLN                * Print a new line

RITE_REG_A0 CMP.B   #$30,       D0          * Check if '0'
            BNE     RITE_REG_A1             * Jump to next test
            MOVE.L  D1,         A0          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE REG A1 CMP.B   #$31,       D0          * Check if '1'
            BNE     RITE_REG_A2             * Jump to next test
            MOVE.L  D1,         A1          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_A2 CMP.B   #$32,       D0          * Check if '0'
            BNE     RITE REG A3             * Jump to next test
            MOVE.L  D1,         A2          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_A3 CMP.B   #$33,       D0          * Check if '0'
            BNE     RITE_REG_A4             * Jump to next test
            MOVE.L  D1,         A3          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_A4 CMP.B   #$34,       D0          * Check if '0'
            BNE     RITE_REG_A5             * Jump to next test
            MOVE.L  D1,         A4          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_A5 CMP.B   #$35,       D0          * Check if '0'
```

```
            BNE     RITE_REG_A6             * Jump to next test
            MOVE.L  D1,         A5          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_A6 CMP.B   #$36,       D0          * Check if '0'
            BNE     RITE_REG_A7             * Jump to next test
            MOVE.L  D1,         A6          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_A7 CMP.B   #$37,       D0          * Check if '0'
            BNE     RITE_REG_E              * Jump to next test
            MOVE.L  D1,         A7          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit


RITE_REG_D  BSR     GET_CHAR                * Get the register # to read
            BSR     PRT_NWLN                * Print a new line
            LEA      NDAT,      A4          * Load new data Prompt
            BSR     PRT_STRG                * Print the message

            BSR     GET_FOUR                * Get the first four data points
            LSL.L   #8,         D1          * Shift the data over to make space
            LSL.L   #8,         D1          * Shift the data over to make space
            BSR     GET FOUR                * Get the latter four data points
            BSR     PRT_NWLN                * Print a new line
            BSR     PRT_NWLN                * Print a new line

RITE_REG_D0 CMP.B   #$30,       D0          * Check if '0'
            BNE     RITE_REG_D1             * Jump to next test
            MOVE.L  D1,         D0          * Move the data to memory
            MOVE.L  (A7)+,      D1          * Restore working register
            MOVE.L  (A7)+,      A4          * Move D0 save to trash
            RTS                             * Return

RITE_REG_D1 CMP.B   #$31,       D0          * Check if '1'
            BNE     RITE_REG_D2             * Jump to next test
            MOVE.L  (A7)+,      D0          * Move D1 save to trash
            MOVE.L  (A7)+,      D0          * Restore working register
            RTS                             * Return

RITE_REG_D2 CMP.B   #$32,       D0          * Check if '0'
            BNE     RITE_REG_D3             * Jump to next test
            MOVE.L  D1,         D2          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_D3 CMP.B   #$33,       D0          * Check if '0'
            BNE     RITE_REG_D4             * Jump to next test
            MOVE.L  D1,         D3          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_D4 CMP.B   #$34,       D0          * Check if '0'
            BNE     RITE_REG_D5             * Jump to next test
            MOVE.L  D1,         D4          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_D5 CMP.B   #$35,       D0          * Check if '0'
            BNE     RITE_REG_D6             * Jump to next test
            MOVE.L  D1,         D5          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE REG D6 CMP.B   #$36,       D0          * Check if '0'
            BNE     RITE_REG_D7             * Jump to next test
            MOVE.L  D1,         D6          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit

RITE_REG_D7 CMP.B   #$37,       D0          * Check if '0'
            BNE     RITE REG E              * Jump to next test
            MOVE.L  D1,         D7          * Move the data to memory
            JMP     RITE_REG_X              * Jump to exit


RITE_REG_E  JMP     PRT_ERR                 * Jump to printing

RITE_REG_X  MOVE.L  (A7)+,      D1          * Restore working register
            MOVE.L  (A7)+,      D0          * Restore working register
            RTS                             * Return


*------------------ RITE_2REC ---------------------------------------
* Purpose: Writes the S-Record to memory
* In:  GET_CHAR, GET_FOUR
* Out: S-Record to RAM
*-------------------------------------------------------------------
RITE_2REC   MOVE.L  D2,         -(A7)       * Store working register
            MOVE.L  D1,         -(A7)       * Store working register
            MOVE.L  D0,         -(A7)       * Store working register

            MOVE.L  #$00000000, D1
RITE_REC_L1 LSL.B   #4,         D1          * Shift to make space for next input
            BSR     GET_CHAR                * Get the next char
            BSR     MAKE_HEX                * Try to convert the input to hex
            SUBI.B  #$01,       D2          * Decrement
            BNE     RITE_REC_L1             * Restart loop if needed
            MOVE.B  D1,         D2

            LSL.B   #4,         D1          * Shift to make space for next input
            BSR     GET_CHAR                * Get the next char
            BSR     MAKE HEX                * Try to convert the input to hex
            LSL.B   #4,         D1          * Shift for rest of byte
            BSR     GET_CHAR                * Get the next character
            BSR     MAKE_HEX                * Convert to hex
            LSL.L   #8,         D1          * Shift over 1 byte
            LSL.L   #8,         D1          * Shift over 1 byte

            BSR     GET_FOUR                * Get target address
            MOVE.W  #$0001,     D0          * Move 1 into D0
            AND.B   D1,         D0          * Isolate last bit of D0
            BNE     PRT_ERR                 * If address is odd, Err


            MOVE.L  D1,         A2          * Move the data to the address register
            SUBI.B  #$04,       D2          * Decrement for used bytes and to ignore checksum
```

```
RITE_REC_L2 LSL.B   #4,         D1          * Shift to make space for next input
            BSR     GET_CHAR                * Get the next char
            BSR     MAKE_HEX                * Try to convert the input to hex
            LSL.B   #4,         D1          * Shift for rest of byte
            BSR     GET_CHAR                * Get the next character
            BSR     MAKE_HEX                * Convert to hex
            MOVE.B  D1,         (A2)+
            SUBI.B  #$01,       D2          * Decrement
            BNE     RITE_REC_L2             * Restart loop if needed
            MOVE.B  D1,         D2

            BSR     GET_CHAR                * Absorb checksum
            BSR     GET_CHAR                * Absorb checksum
            BSR     GET_CHAR                * Absorb CR


            MOVE.L  (A7)+,      D0          * Restore working register
            MOVE.L  (A7)+,      D1          * Restore working register
            MOVE.L  (A7)+,      D2          * Restore working register


            RTS                             * Else return

*------------------ RITE_8REC ---------------------------------------
* Purpose: Writes the S-Record to memory
* In:  GET_CHAR, GET_FOUR
* Out: PRT_****, Write to memory
*-------------------------------------------------------------------
RITE 8REC   MOVE.L  D2,         -(A7)       * Store working register
            MOVE.L  D1,         -(A7)       * Store working register
            MOVE.L  D0,         -(A7)       * Store working register


RITE_REC_L8 LSL.B   #4,         D1          * Shift to make space for next input
            BSR     GET_CHAR                * Get the next char
            BSR     MAKE_HEX                * Try to convert the input to hex
            SUBI.B  #$01,       D2          * Decrement
            BNE     RITE_REC_L8             * Restart loop if needed
            MOVE.B  D1,         D2

            LSL.B   #4,         D1          * Shift to make space for next input
            BSR     GET_CHAR                * Get the next char
            BSR     MAKE_HEX                * Try to convert the input to hex
            LSL.B   #4,         D1          * Shift for rest of byte
            BSR     GET_CHAR                * Get the next character
            BSR     MAKE_HEX                * Convert to hex
            LSL.L   #8,         D1          * Shift over 1 byte
            LSL.L   #8,         D1          * Shift over 1 byte

            BSR     GET_FOUR                * Get target address
            MOVE.W  #$0001,     D0          * Move 1 into D0
            AND.B   D1,         D0          * Isolate last bit of D0
            BNE     PRT_ERR                 * If address is odd, Err

            MOVE.L  D1,         A1          * Move the data to the address register
            SUBI.B  #$04,       D2          * Decrement for used bytes and to ignore checksum

            BSR     GET_CHAR                * Absorb checksum
            BSR     GET_CHAR                * Absorb checksum

            BSR     PRT_NWLN                * Print new line
            BSR     PRT_NWLN                * Print new line


            MOVE.L  (A7)+,      D0          * Restore working register
            MOVE.L  (A7)+,      D1          * Restore working register
            MOVE.L  (A7)+,      D2          * Restore working register


            RTS                             * Else return

*------------------ RITE_MREC ---------------------------------------
* Purpose: Writes the S-Record to memory
* In:  GET_CHAR
* Out: PRT_****, RITE_*REC
*-------------------------------------------------------------------
RITE_MREC
            MOVE.L  D2,         -(A7)       * Store working register
            MOVE.L  D1,         -(A7)       * Store working register
            MOVE.L  D0,         -(A7)       * Store working register
            BSR     PRT_NWLN                * Print new line
            LEA     _RREC,      A4          * Load S-Record Prompt
            BSR     PRT_STRG                * Print it


RITE_MREC_L MOVE.B  #$02,       D2          * Initialize the counter

            BSR     GET_CHAR                * Get first character
            CMP.B   #$53,       D0          * Check if 'S'
            BEQ     RITE REC LS
            CMP.B   #$73,       D0          * Check if 's'
            BEQ     RITE_REC_LS
            JMP     PRT_ERR                 * Else error and return

RITE_REC_LS BSR     GET_CHAR                * Get second character
            CMP.B   #$32,       D0          * Check if '2'
            BEQ     RITE_MREC1              * Get Data
            CMP.B   #$38,       D0          * Check if '8'
            BEQ     RITE_MREC2
            JMP     PRT_ERR                 * Else error and return

RITE MREC1  BSR     RITE 2REC
            JMP     RITE_MREC_L

RITE_MREC2  BSR     RITE_8REC

            MOVE.L  (A7)+,      D0          * Restore working register
            MOVE.L  (A7)+,      D1          * Restore working register
            MOVE.L  (A7)+,      D2          * Restore working register
```

```
           RTS                         * Else return

*------------------ RUN_REC ---------------------------------------
* Purpose: Runs the S-Record in memory
* In:  N/A
* Out: N/A
*-----------------------------------------------------------------
RUN_REC    JSR     (A1)
           RTS

*------------------ RUN_OPT ---------------------------------------
* Purpose: Calls the subroutine the user has selected
* In:  D0
* Out: NA
* Note: A4 overwritten
*-----------------------------------------------------------------
RUN_OPT    CMPI.B  #$31,       D0      * If option 1 is not selected
           BNE     RUN_OPT_2           * Go to option 2
           BSR     READ_ROM            * Else read from ROM
           RTS                         * Return

RUN_OPT_2  CMPI.B  #$32,       D0      * If option 2 is not selected
           BNE     RUN OPT 3           * Go to option 3
           BSR     READ_RAM            * Else read from RAM
           RTS                         * Return

RUN_OPT_3  CMPI.B  #$33,       D0      * If option 3 is not selected
           BNE     RUN_OPT_4           * Go to option 4
           BSR     READ REG            * Else read from Register
           RTS                         * Return

RUN_OPT_4  CMPI.B  #$34,       D0      * If option 4 is not selected
           BNE     RUN_OPT_5           * Go to option 5
           BSR     RITE_RAM            * Else write RAM
           RTS                         * Return

RUN_OPT_5  CMPI.B  #$35,       D0      * If option 5 is not selected
           BNE     RUN_OPT_6           * Go to option 6
           BSR     RITE_REG            * Else write to a register
           RTS                         * Return

RUN_OPT_6  CMPI.B  #$36,       D0      * If option 6 is not selected
           BNE     RUN_OPT_7           * Go to option 7
           BSR     RITE_MREC           * Else Write the S-Record
           RTS                         * Return

RUN_OPT_7  CMPI.B  #$37,       D0      * If option 6 is not selected
           BNE     RUN_OPT_E           * Go to error state
           BSR     RUN_REC             * Else run S-Record
           RTS                         * Return



RUN_OPT_E  JMP     PRT_ERR             * Error and reset

           SIMHALT                     * Halt simulator
           END     BEGIN               * last line of source
```

Troubleshooting

Hardware

Socket Sizes

The first issue discovered while making this project was the realization that the CPLD place on the PCB had been designed for a 68-pin IC rather than the 84-pin IC that had been purchased. Additionally, the RAM chips were too wide for their position on the board. To solve this problem, perf-board was used to create adapters between the PCB and the sockets, using solder bridges to connect the pins on the socket with header pins spread out to the necessary locations.

Pin-Layout Error

The CPLD required further work as there is no way to adapt 84 pins to fit into 68 holes, thus braided copper wire was purchased with female headers which could be run from the CPLD breakout board to the PCB in any configuration needed. This method also allowed for changes to be made as later issues were discovered, without the need to resolder connections.

Poor CPU Socket

The CPU socket purchased did not hold the CPU well enough to ensure stable connection. The connection was so poor, the CPU would dislodge from its position after nearly every test. This was resolved by soldering the chip directly to the socket.

Missing Clock Trace

The CPU Clock trace was missing its final segment, whether due to damage during soldering or a flaw in the board during manufacturing. The schematic shows the trace connected across the missing section. This was resolved by applying a solder bridge across the missing trace.

Damaged CPLD Clock

While the CPLD Clock seemed to be functioning well according to the logic analyzer, a test with the oscilloscope showed it alternated between 2.2V and 2.7V rather than 0V and 5V. Replacing this clock showed that the spare was also malfunctioning. This was ultimately solved by running the CPLD off the clock intended for the DUART.

Electromagnetic Interference From Clocks

After changing the clocks, there was still periodic interruptions in the monitor program. This was decided to be due to the clocks and their proximity to power lines. This was resolved by moving the clocks to an external breadboard.

Poor Power Lines

Due to continuing interruptions in the monitor program, it was decided there was not enough power reaching the CPLD. This was resolved by changing from braided wire to solid core.

Failing Pull-Up Of DUART DTACK

The DTACK pin of the DUART, when probed, showed it was failing to go high when the DUART did not have data with which to interact. This was tied high with a 1kΩ resistor to solve the problem.

Poor Socket Connection To DUART

Constant probing of the DUART damaged the pins in the socket. This was resolved by inserting bare wires between the damaged pins and the DUART to reconnect the pins.

Software

Write To RAM Failure

When the monitor program was tested, the Write To RAM feature was continually failing. The error was due to a register being restored at the end of a sub-procedure and overwriting the calculated data. A change of this code resolved the issue.

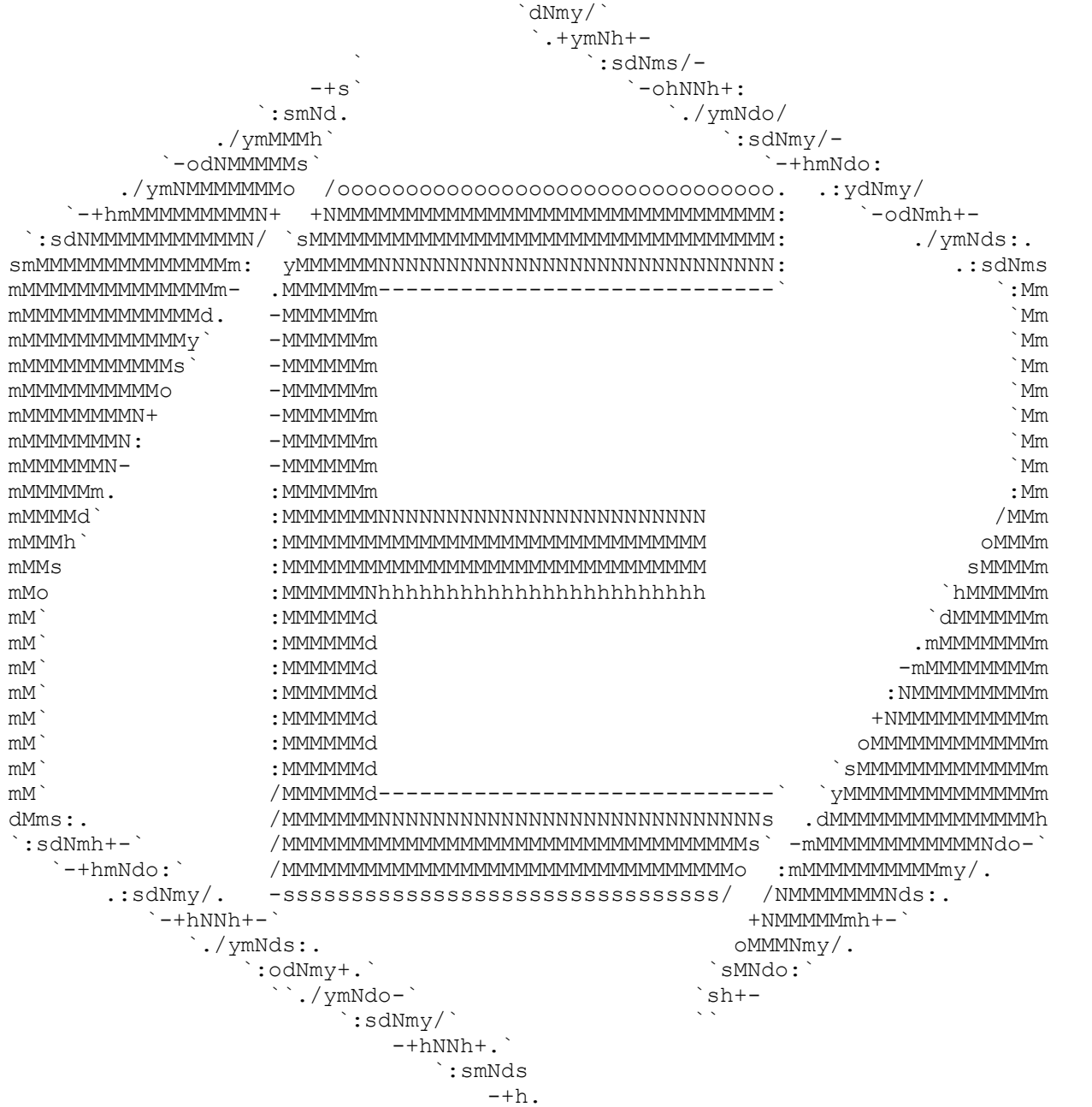
Conclusion

While there were many hours of debugging necessary to bring this project to the desired level of functionality, it was a useful project for understanding the intricacies necessitated by computers. I have a deeper appreciation for the complexity of modern devices and a better understanding of how they function. This knowledge will be useful in my work as a software engineer as I now better understand the methods used to enable the higher-level code I write. Further, I have learned many useful techniques in debugging a system without direct visual information; specifically, the use of multimeters, digital logic analyzers, and oscilloscopes, each as required by the current issue.

Appendix A - Demonstration Output

```
Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 1
Please enter a valid 2 byte address: 1234
Address Accepted!

The memory contents at your selected index are: $67066100


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 2
Please enter a valid 2 byte address: 1234
Address Accepted!

The memory contents at your selected index are: $F9AF977C


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 4
Please enter a valid 2 byte address: 1234
Address Accepted!

Please enter the new data for this address: $12341234


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 2
Please enter a valid 2 byte address: 1234
Address Accepted!

The memory contents at your selected index are: $12341234


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
```

```
7) RUN   the  S-Record
Eklektik@uComp:~$ 3
Please select a register (A0-A7, D0-D7): d6
The memory contents at your selected index are: $FFFFFFFF


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 5
Please select a register (A0-A7, D0-D7): d6
Please enter the new data for this address: $12341234


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 3
Please select a register (A0-A7, D0-D7): d6
The memory contents at your selected index are: $12341234


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 6
Please enter the S-record:
S224F02100200D0A2020202020202020202020202020202020202020S804F03004D7FFCB0449F900F021
004EB812304E75FFFF492D2B682E0D0A005F202020202049


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$
The input you submitted is invalid in this context.


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 7
                                    `o-
```



```
Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$

Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 1
Please enter a valid 2 byte address: 1233
The input you submitted is invalid in this context.


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
```

```
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 3
Please select a register (A0-A7, D0-D7): w
The input you submitted is invalid in this context.


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 6
Please enter the S-record: 5
The input you submitted is invalid in this context.


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$ 6
Please enter the S-record: s205t
The input you submitted is invalid in this context.


Welcome to the Eklektik Design uComputer Monitor Program
Select an option below:
1) READ  from ROM
2) READ  from RAM
3) READ  from Register
4) WRITE to   RAM
5) WRITE to   Register
6) WRITE to   S-Record
7) RUN   the  S-Record
Eklektik@uComp:~$
```