Microcomputer Design
Final Report

Presented to

Chandler Griscom
chandlergriscom@letu.edu

Presented to

Dr. Joonwan Kim
EEGR 4253
23 Apr 2017

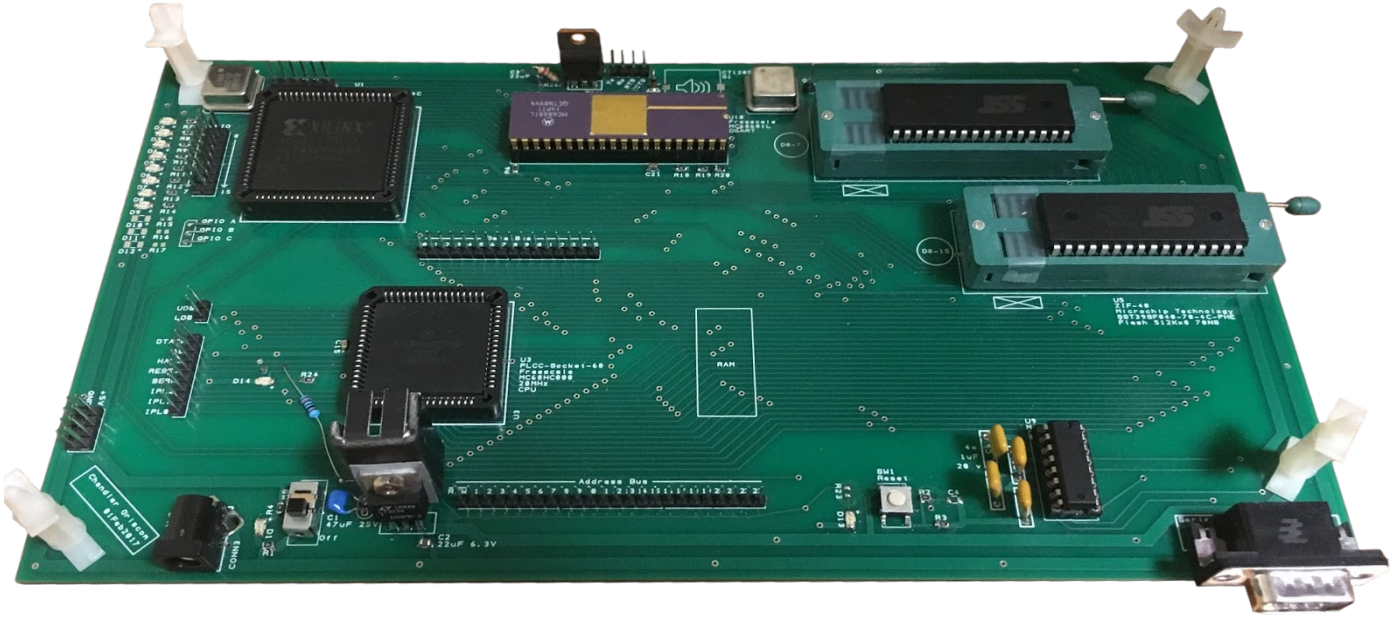
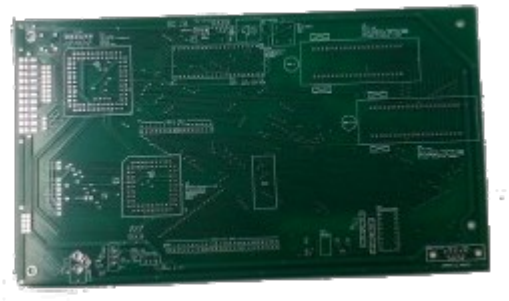


Table of Contents

I. Description of Project

Motorola 68000 Single-Board Computer Project

This project was designed and built for EEGR4253 – Microcomputer Design. In that course, students are required to design from scratch and build a working computer with the following minimum hardware requirements:

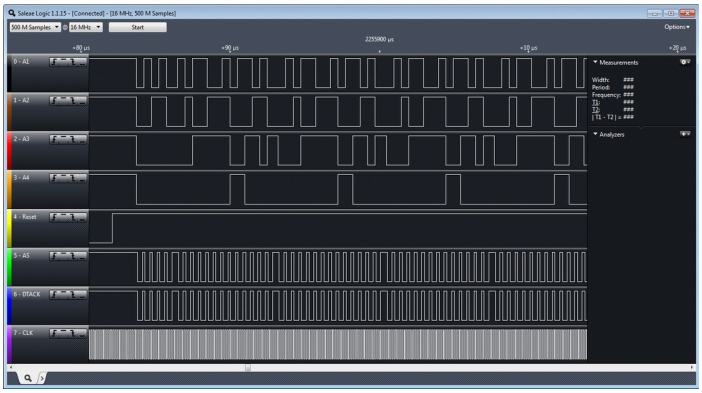


- An RS-232 compatible data port capable of operating at 19,200 baud
- 64K of RAM (random access memory)
- The capability of addressing 64K of memory space.
- An “on-board” monitor program residing in EPROM, EEPROM, or any non-volatile memory.

Additionally, the computer is required to support the following software operations:

- 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 all CPU registers
- Transmitting and receiving data via all data ports
- Allowing a machine-coded program to be loaded into RAM from a data port
- Executing a program that has been loaded into RAM

This project was completed by Chandler Griscom over the course of Spring 2017 semester at LeTourneau University. Skills acquired as a result of completing the project include:



- Implementing logical constructs using a Xilinx CPLD and VHDL
- Programming in 68k assembly language
- Debugging hardware with a logic analyzer
- Understanding of the low-level hardware organization of a computer
- Understanding of basic operating system concepts

The remainder of this report is broken up into four sections: hardware details, software details, troubleshooting accounts, and an appendix containing assembly code listings.

II. Hardware Description

1. Introduction

The hardware selection and design decisions for this 68k single board computer were made based on recommended parts, availability of obsolete components, pricing, and prospects of future usage. In addition to the core project requirements, the following components and connections were deemed useful:

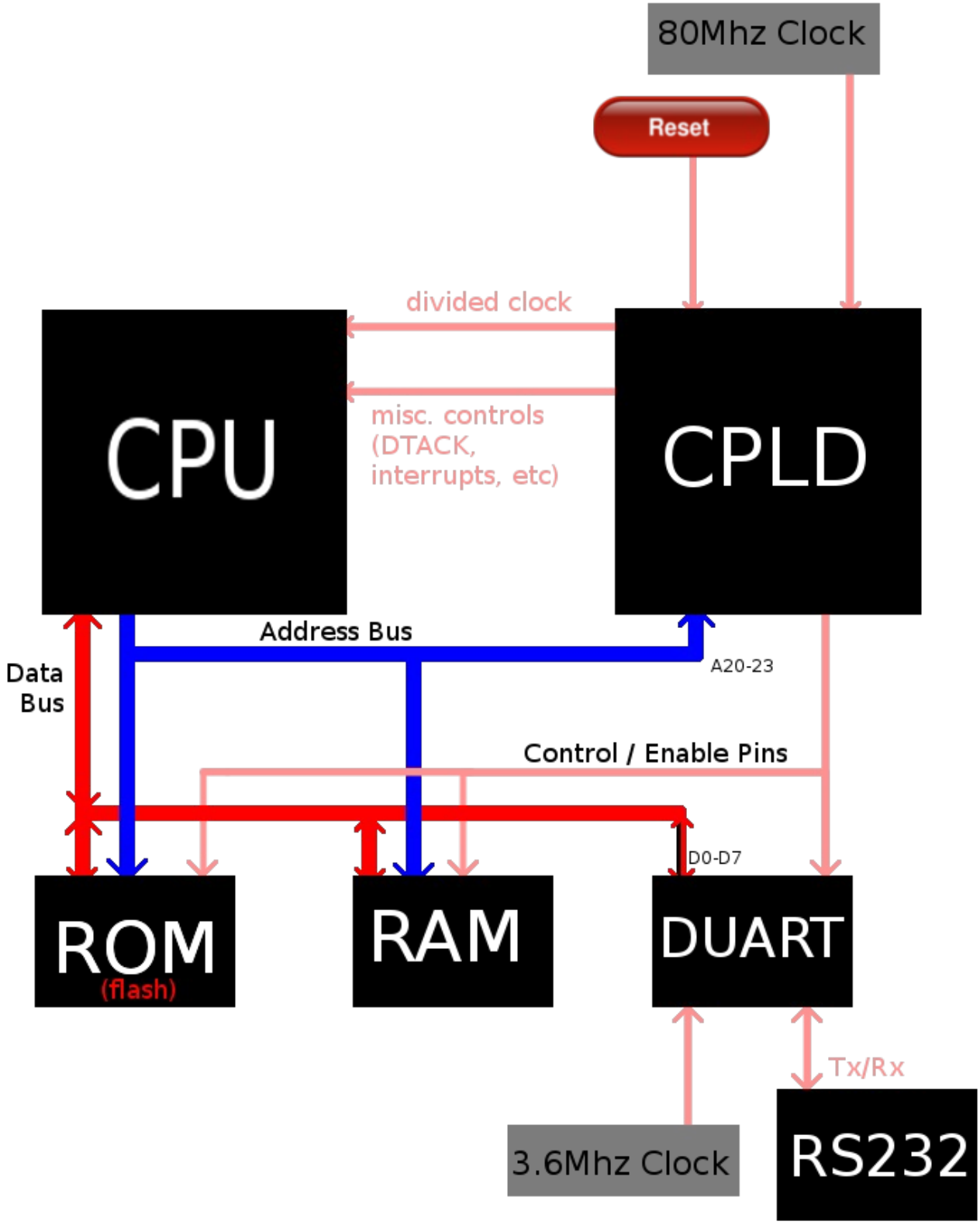
- Both RS232-compatible and TTL (i.e. Arduino) compatible serial ports
- Writable flash memory that doubles as ROM – should be editable without lab equipment
- Interrupt lines for implementing a timer (needed for preemptive scheduling and possibly for writing to flash memory)
- A dividable clock (80 MHz may be divided into 8, 10, or 20 MHz)
- A buzzer/speaker driver (these parts were never ordered but there is space on the board)
- ZIF, DIP, and PLCC sockets for easily switching out components (especially flash chips)

Much of the necessary research was able to be postponed by routing “unknown” connections to the XC9572 CPLD and then later internally mapping these connections using VHDL code.

The PCBArtist application was used to design the 2-layer circuit board for the project. All component datasheets were analyzed and drawn up in the software. A schematic of the electrical connections was created and then translated to a printed circuit board layout by manually orienting components and drawing copper traces.

The finished circuit board design was ordered from 4pcb under their student discount.

2. Block Diagram

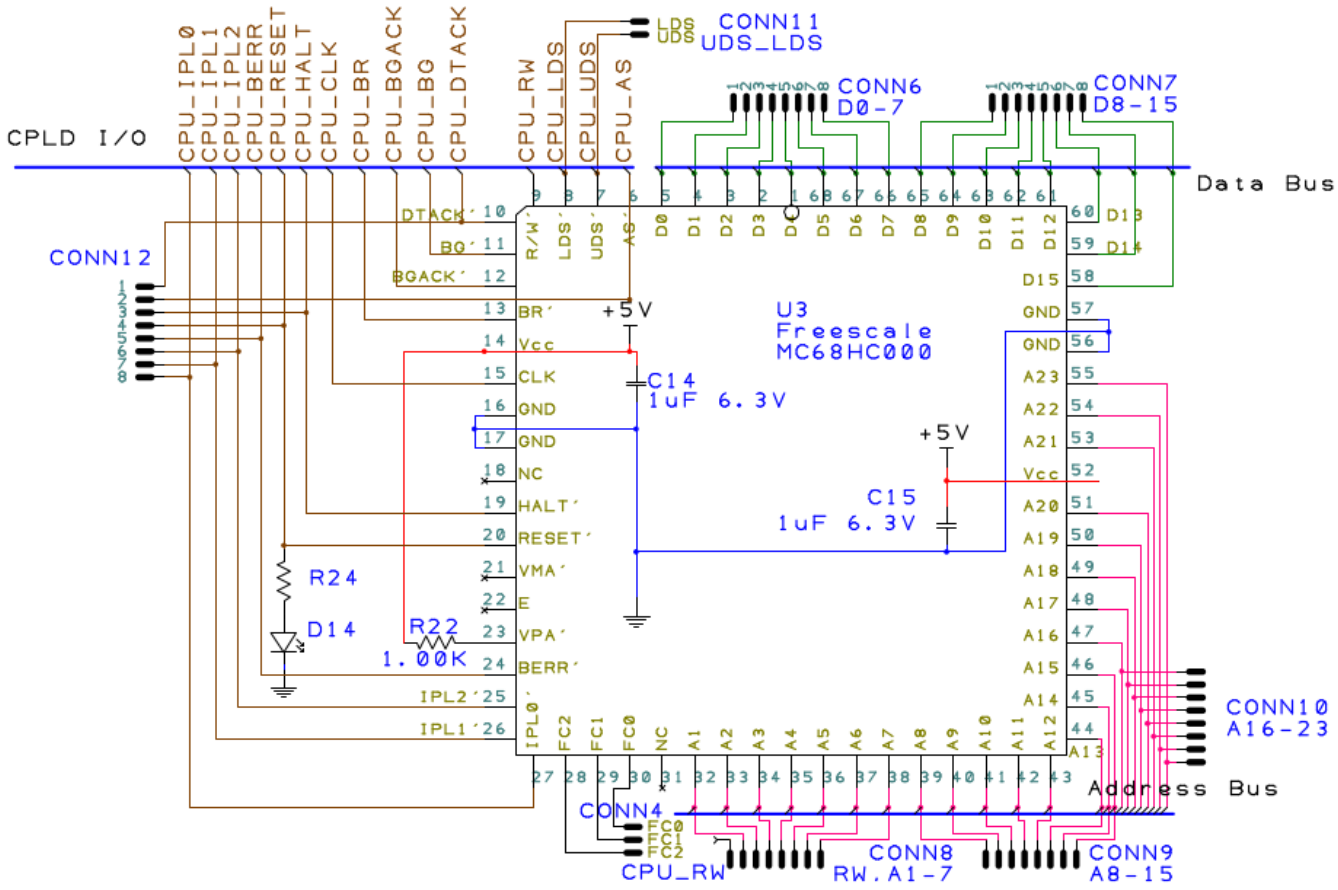


3. Technical Specifications

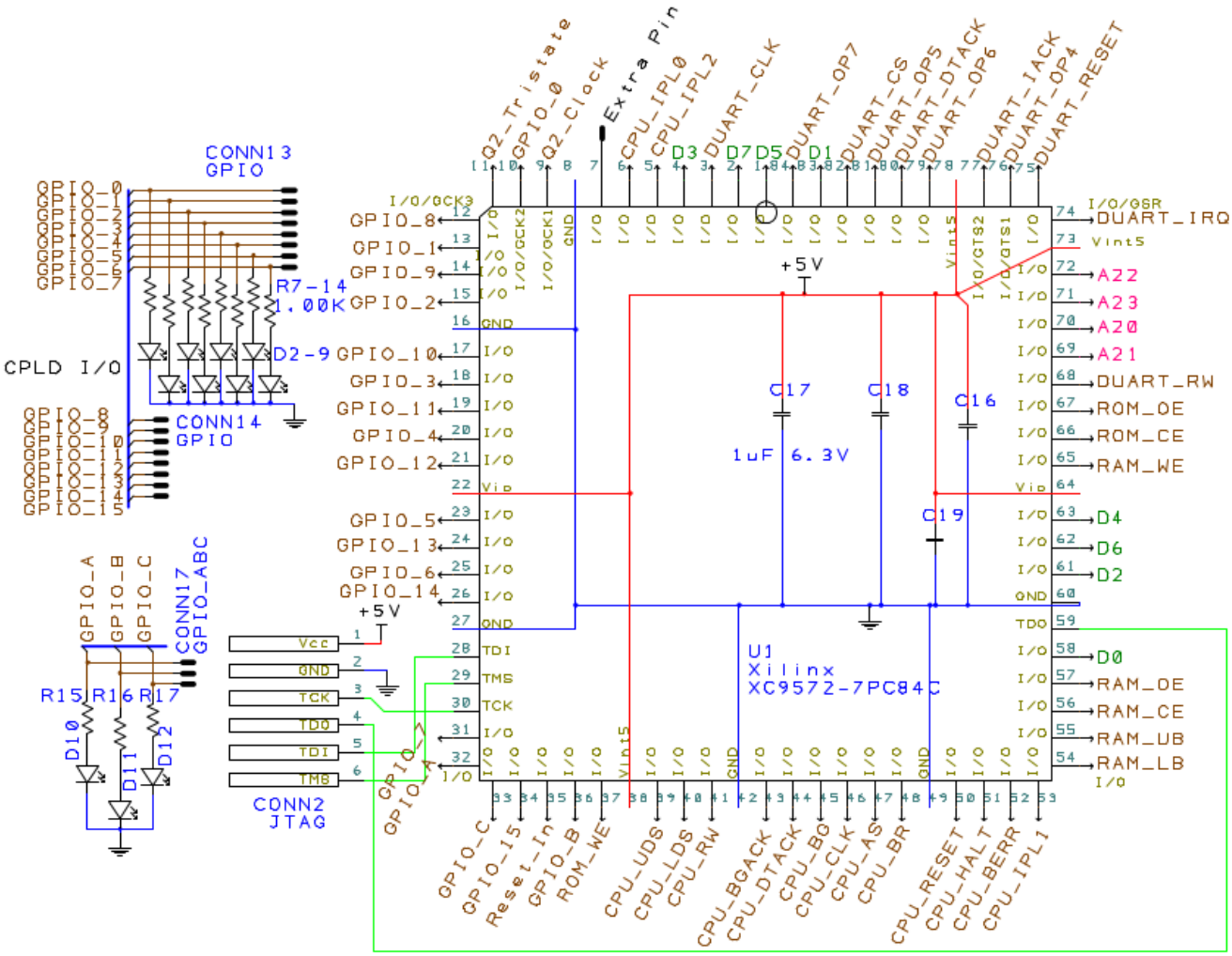
- Motorola 68000 Processor – Clocked at 20MHz (Freescale MC68HC000FN20)
- ½ MB Static RAM (Alliance Memory AS7C4098A-12JIN)
- 1 MB Flash Memory (2x Microchip Technology SST39SF040-70-4C)
- 16 general purpose buffered I/O pins (including 8 LEDs)
- DUART: 2 serial ports at 19200 baud rate (Motorola MC68681L)
  - Port A: RS232-compatible (10 V), (driven by Texas Instruments MAX232EIN)
  - Port B: TTL-compatible (5 V)
- CPLD providing a 1 ms timer, glue logic, interrupt generators (Xilinx XC9572-7PC84C)
- 10.5” x 5.5” 2-layer PCB (ordered from 4pcb)

4. Schematics

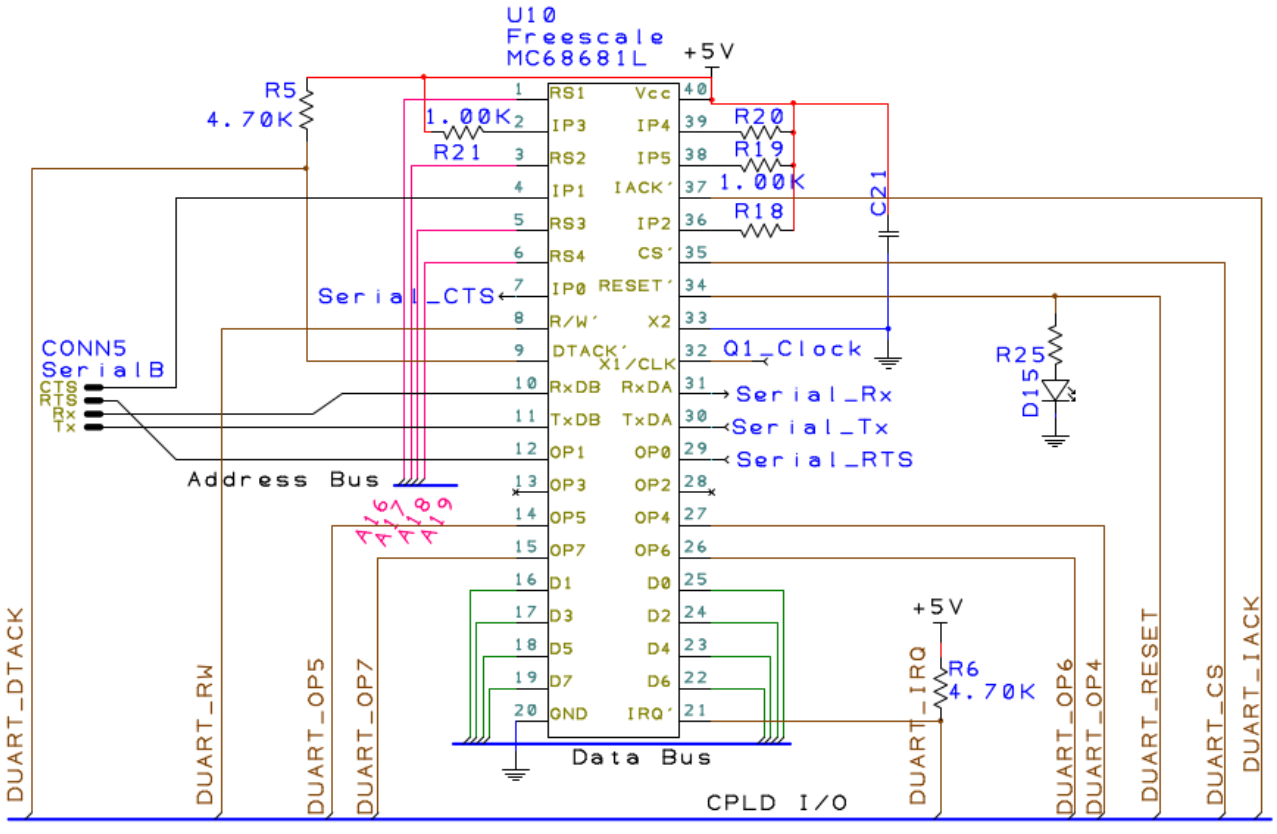
a. CPU Connections



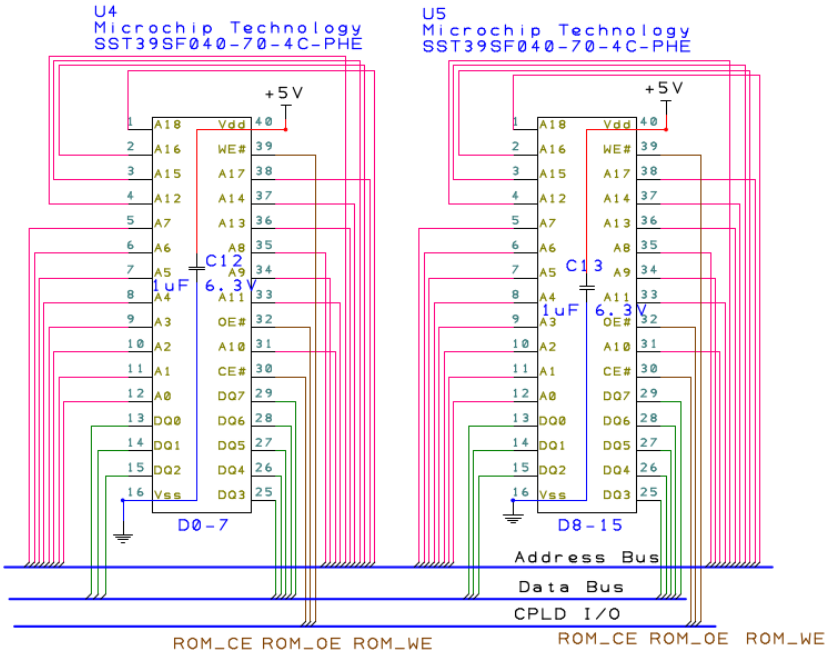
b. CPLD Connections



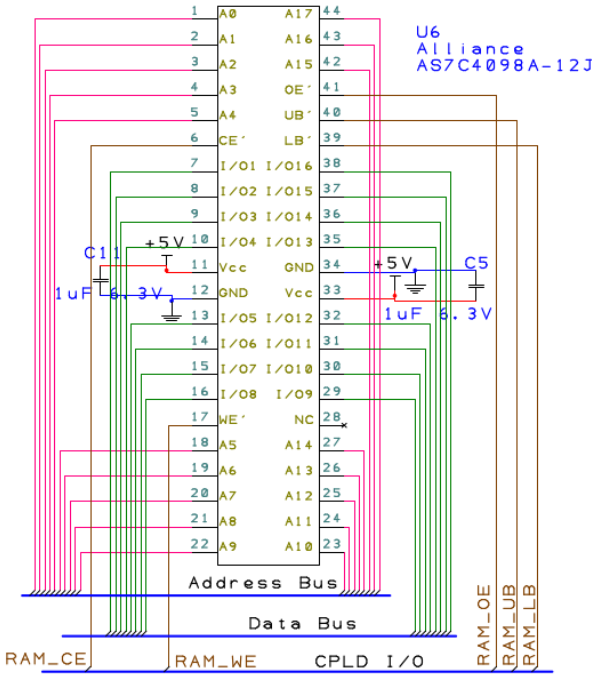
c. DUART Connections



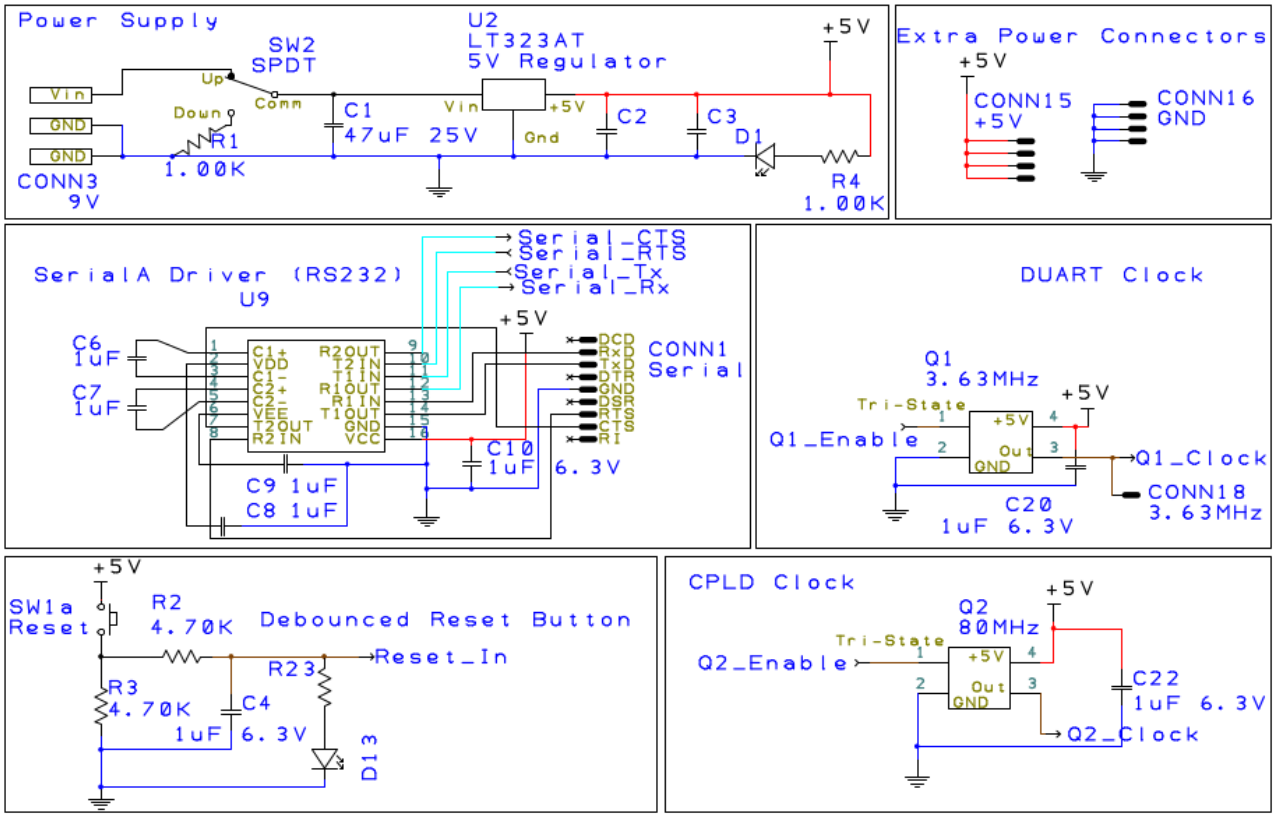
d. ROM Connections



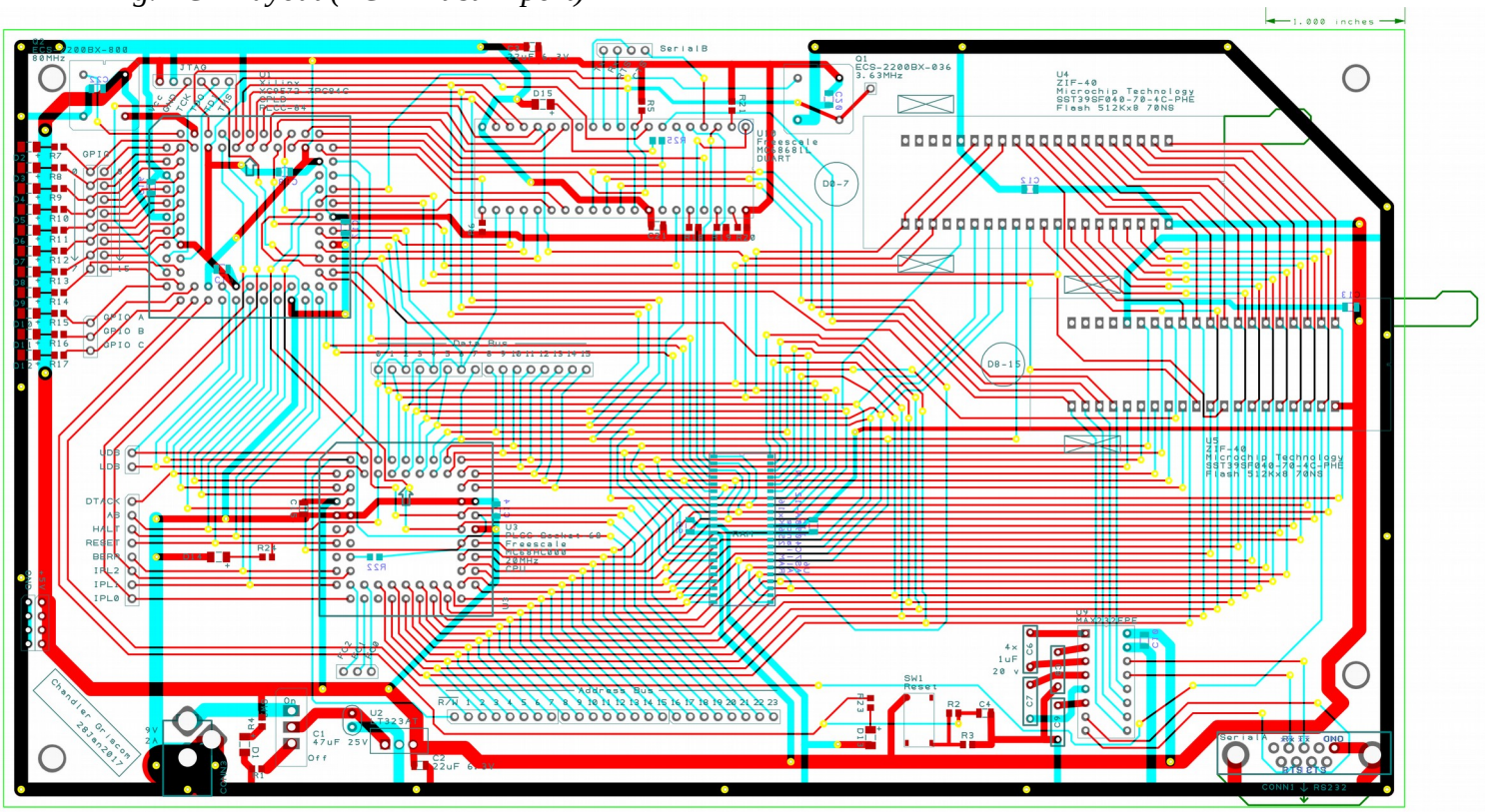
e. RAM Connections



f. Miscellaneous Connections



g. PCB Layout (PCB Artist Export)



5. Price Figures

Item	#	Price(sum)	Description
Processor (eBay)	5	$11.28	68000 CPU 20MHz, 68-pin PLCC
Processor Socket	2	$2.84	CONN SOCKET PLCC 68POS TIN
CPU Oscillator	1	$2.37	OSC XO 80.000MHZ TTL PC PIN
Oscillator Sockets	2	$0.50	8-pin dip socket
CPLD (eBay)	2	$16.00	IC CPLD 72 MCELL C-TEMP 84-PLCC
CPLD Socket	5	$2.56	5x PLCC IC Socket DIP 84 PINS PLCC
Pin Headers	3	$2.04	Headers & Wire Housings 16CKT
DUART (eBay)	2	$9.99	40-DIP
Duart Socket	1	$1.02	CONN SOCKET PLCC 44POS TIN
Duart CLK	1	$1.88	OSC 3.6864 MHZ 5.0V FULL SIZE
RS232 Driver	2	$2.74	IC DUAL RS232 ESD-PROT 16-DIP
RS232 Driver Socket	1	$0.50	16-pin dip socket
D-Sub connector	1	$1.54	CONN D-SUB PLUG 9POS R/A
Ram	2	$9.96	IC SRAM 4MBIT 12NS 44SOJ
Rom	4	$6.68	IC FLASH 4MBIT 70NS 32DIP
Rom ZIF Socket (eBay)	2	$4.98	40Pin ZIF ZIP DIP
5v Regulator	1	$4.27	IC REG LDO 5V 3A TO220-3
ON/OFF switch	2	$2.18	Slide Switches MINI SLIDE SWITCH
Reset Button	2	$1.52	SWITCH TACT 6MM SMD MOM 160GF
LEDs	18	$5.63	SMD 3.2X1.6MM LOW CRNT Blue
Resistor 1.0k	100	$0.60	SMD 0805 1Kohms 5% AEC-Q200
Resistor 4.7k	100	$0.60	SMD 0805 4.7Kohms 5% AEC-Q200
Capacitor 1 uF	35	$1.33	SMD 0805 1uF 16volts X5R 10%
Capacitor 22 uF 6.3V	2	$0.48	SMD 0805 22uF 6.3volts X5R 20%
Capacitor 47 uF 25V	1	$0.99	Ceramic 47uF 25volts X5R 20%
Barrel Plug (Male)	1	$0.85	CONN PWR JACK 2.1X5.5MM SOLDER
9V AC Adaptor	1	$6.59	9V 2A 2000mA DC AC Adapter
Logic Analyser	1	$6.79	
PCB Board	1	$33.00	
PCB Shipping	1	$22.00	
Digikey Shipping	1	$10.55	
Mouser Shipping+Tax	1	$7.77	
68000 Shipping	1	$2.00	
DUART Shipping	1	$2.50	
CPLD Shipping	1	$4.10	
84pin socket shipping	1	$1.85	
ZIF socket shipping	1	$2.54	
TOTAL with shipping:		**$193.02**	

6. Time Figures

Time Sheet	Hours	
Break	3	Researched and bought CPLD, 68k, CPLD socket from eBay
Week 1	2	Selecting and ordering parts
	2.5	Selecting and ordering parts
	1	Attend Friday Lab Meeting and work on prelim schematic
Week 2	1	Finish Prelim Schematic, write progress report
	4.5	Completed ALL component selection. Ordered DUART, AC adaptor, serial cables from eBay & Amazon
	2	Switch out some resistors & capacitors, place Mouser and Digikey orders
	4.5	Download and draw symbols
	2	Finish drawing symbols
Week 3	2	Begin Schematic – RS232, Reset Debounce, Regulator
	9	Work on Schematic
	1	Lab – Talk to josh about block diagram
	1	Explore DUART functionality
	3	Edit schematic (add 80% of testpoints, remove unneeded connections)
	2	Consult Josh during lab regarding schematic, soldering, DUART, work on block diagram
	3	Work on Schematic
	1	Start researching and laying out PCB
	6	PCB Layout (assemble busses, connect most chips)
	5	PCB Layout (route CPLD)
	2	PCB Layout (sync CPLD with schematic, add capacitors)
	3	Add labels, more LEDS
Week 4	2	Edit schematic and PCB; change WE, CE, OE busses
	2.5	Make schematic look good, make CPLD netlist, final schematic export
	2	Lab with Josh, fit components to printout, fix power jack mistake
	1	Add speaker to unused CPLD pin
	1	Verify parts against datasheets
	1	Order board!!!
	1	Begin VHDL code (netlist, constraints file)
	3	Work on VHDL (read/write logic)
	7	Finish most of VHDL (implemented all chip logic, dtack, interrupts, gpio)
Week 5	0.5	Redo schematic 11x17
	0.5	Go to lab, write progress report
	1	Setup pulseview/sigrok for saleae logic analyser
	0.5	Test logic analyser using 555 timer (success!)
	3	Solder all but RAM
	4	Test VHDL
	5	Get ROM Working
Week 6	5	Debug sporadic issues caused by lack of BGACK
	0.5	Got RAM working!
	4	Working periodic (IPL7) interrupts
	4	Duart loopback test is working
	5	Working duart communication (polling)
	0.25	Working duart (IPL6) interrupts!
	4	Write and debug hello world program
	6	Program 'bootloader' with S record upload capability
	2	Write Bash script for sending S records to bootloader
	2	Update clock to 20Mhz, add heatsink
	3	Debug and fix DUART interrupts
	5	Work on / research ROM burning from the CPU. Started monitor program
	3	Write VHDL modifications and 68k code for hotswapping ROM chips
Week 7	7	Finished ROM burner – can now overwrite boot record
	0.5	Establish DUART channel B communication
Week 8	12	Add ROM-writing functionality to Monitor program for uploading S Records
Week 9	4	Finish monitor program (view/edit memory and registers)
	0.5	Write RC Car S Record
Week 10	0.5	Presentation in Digital Class
Total	163.75	hours

III. Software Description

1. Introduction

The software for the 68000 single board computer is comprised of a Monitor Program (essentially a simple operating system), a loadable ROM burner program, and uploadable S records (ASCII-encoded machine code). It was developed incrementally by first implementing an S-record "bootloader" which allowed complex programs to be tested in RAM, then creating a ROM burner program that could overwrite the bootloader, and finally the Monitor Program itself. This incremental development was done so that most of the testing could be done outside of the Microcomputer Design lab once the S record routines were functioning.

I used the software portion of the project to test out some concepts that had been studying for COSC3503 – Operating Systems. The interrupt handling, DUART buffering, and ROM management subsystems were all motivated and inspired by topics from that course. Although they were not required for Microcomputer Design, they provided experience that supplemented my learning in both classes.

2. Memory Mapping and Interrupt Details

a. Memory Space Organization

Address MSB	A23	A22	A21	A20	MODE	Comments
8 through F	1	x	x	x	GPI/GPO	Writes to GPIO 0-7 (LEDs), reads from GPIO 8-15
4 through 7	0	1	x	x	Unused	
3	0	0	1	1	CPLD Mode	Bit 2: Reset DUART, Bit 3: Enable GPIO Interrupts, Bit 4: Pause CPU
2	0	0	1	0	DUART	A19-16 used for register-select
1	0	0	0	1	RAM	512KB addressable; A19 is a don't-care
0	0	0	0	0	ROM	1 MB addressable

b. *Interrupt Levels and Vectors*

Interrupt	Level	IPL2	IPL1	IPL0	Interrupt Vector
	7	1	1	1	
1000Hz Timer	6	1	1	0	65 ($101 hex)
	5	1	0	1	
DUART Interrupt	4	1	0	0	66 ($102 hex)
	3	0	1	1	
	2	0	1	0	
GPIO Interrupt	1	0	0	1	64 ($100 hex)
No Interrupt	0	0	0	0	

3. Core Functionality

Externally, the monitor program is organized as a pair of menus in which options are selected by sending ASCII characters through the serial terminal. It can perform the following operations:

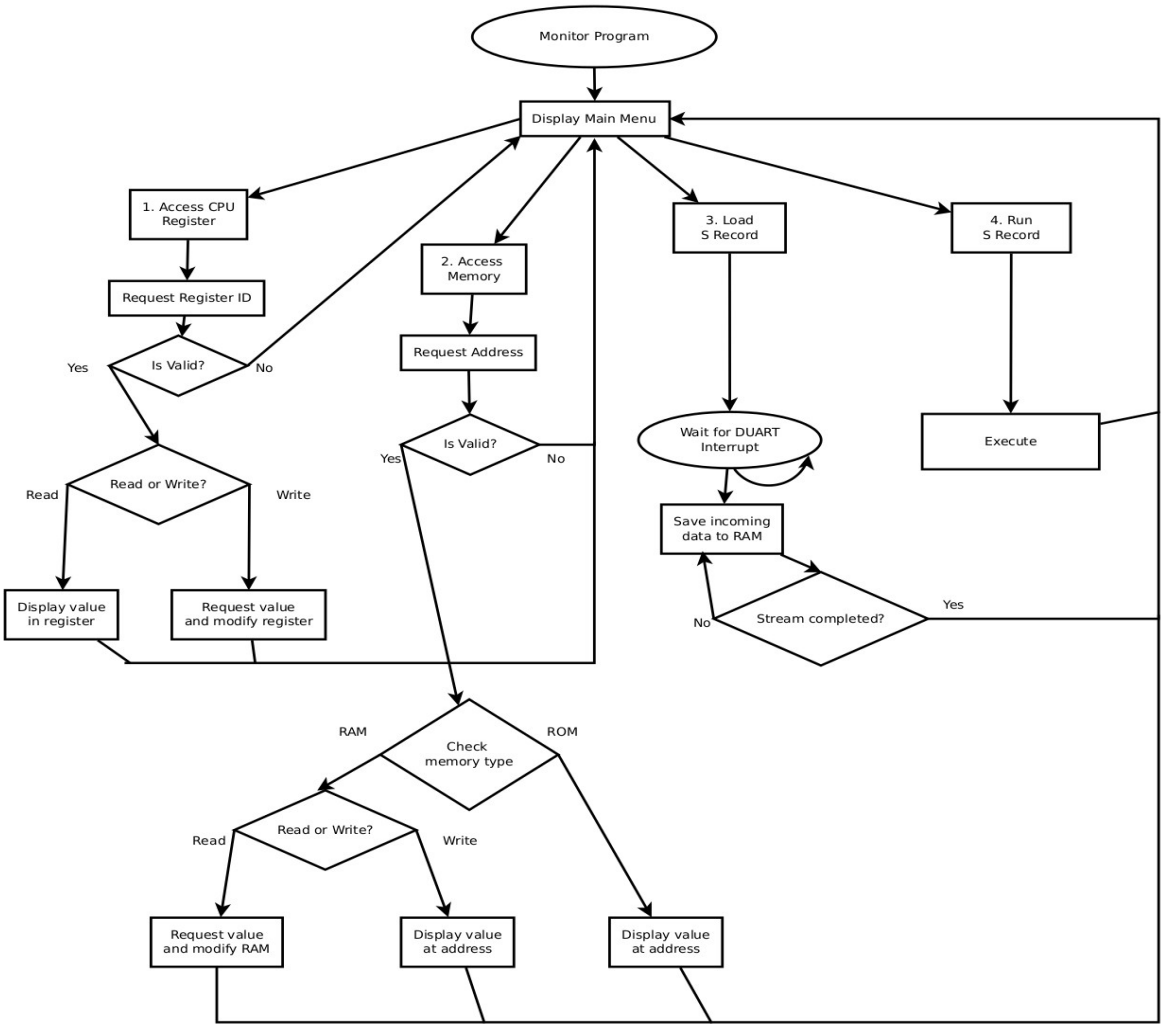
a) Examine and change the contents of any memory location within the memory space
b) Print a 16x16 table of memory locations and values
c) Examine and change the contents of any of the CPU's internal registers
d) Allow a S record-encoded program to be loaded into RAM and executed from the serial port (hidden, for testing purposes)
e) Allow an S record to be copied to ROM and later loaded into RAM for execution
f) Manage settings such as interrupt usage and the LED timer
g) ROM Burning features:
    i. Copy one flash ROM chip to another
    ii. Allow hotswapping chips by temporarily pausing the CPU and DUART
    iii. Erase the entire flash contents and upload a new S record (i.e. overwrite the monitor program)

Internally, the monitor program is comprised of 6 interacting components.

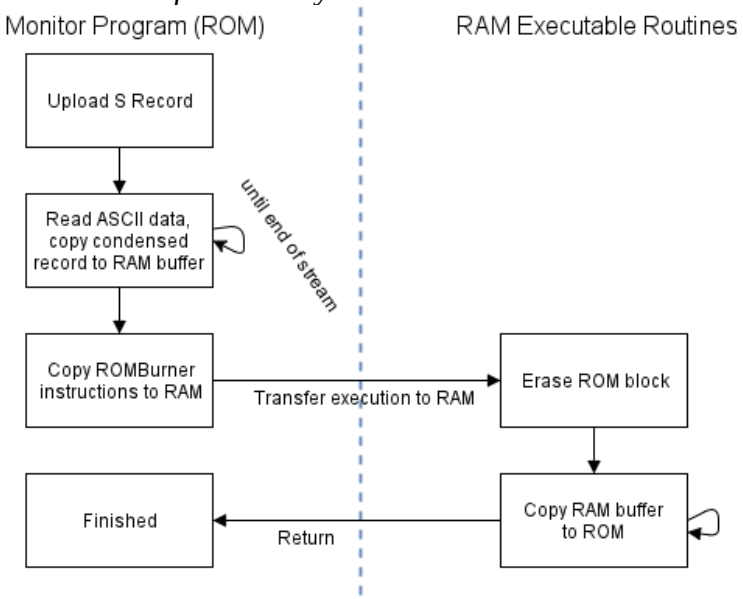
a) Interrupt handlers
b) Menu system
c) ASCII and S record parsing routines
d) S record loading routines (ROM and RAM variants)
e) A loadable ROM burner S record (executes in RAM)
f) A loadable "Copy RAM Buffer to ROM" routine

4. Flowcharts

a. Monitor Program



b. ROM Upload Subsystem



IV. Troubleshooting

1. Hardware Issues

The first issue encountered involved the HALT and RESET pins of the CPU. My schematic treated them as if they were typical inputs, but the datasheet specifies that they may be used as inputs or outputs depending on bus control parameters. If the CPU ever decides to halt itself, inconsistent states / shorts can occur if the CPLD is trying to force HALT or RESET against the CPU output. To fix this I changed the CPLD behavior to treat these two pins as either high-impedance or '0' (active-low) and then added two 4.7K resistors to pull them high by default.

The next notable issue involved a simple misunderstanding on my part; I did not realize that the function code outputs of the CPU were necessary for interrupt acknowledgments and accidentally left them disconnected from the CPLD. Luckily I had exactly three extra CPLD pins and managed to tie FC0, FC1, and FC2 to those pins using jumper wires.

Later on in the project I encountered an issue with one of the PLCC sockets; a single pin connecting the CPLD to the ROM's R/W line became intermittently connected; sometimes it would function as normal, sometimes it wouldn't work at all (preventing the ROM burner from working), and sometimes it would work if I flexed the board. I eventually solved the issue by cleaning the CPLD socket and bending all of the pins out a bit to achieve a better connection.

2. Software Issues

The most evasive and confusing issue I encountered was a simple VHDL mistake; I forgot to tie the unused BERR (bus arbitration error input) pin high. The board experienced issues where the CPU would run a ROM test for about 100ms, then “slow down” (miss clock cycles) and halt itself. Blowing air on a certain area of the board would keep it running for a longer time, leading me to believe it was an issue with overheating. I eventually figured out that it was caused by the indeterminate state of that one input pin.

The counter-intuitive nature of the 68k's data bus caused some issues with the DUART code initially; even addresses use the high byte (D8-15) and odd ones use the low byte (D0-7). My hardware configuration connected the DUART to the low byte, and I did not realize at first that all the DUART register addresses must be odd.

5. Appendix

1. Copy-to-ROM Routine (included as a loadable machine code listing at the end of Monitor Program)

```
*-----------------------------------------------------------
* Title       : Copy RAM Buffer to ROM
* Written by  : Chandler Griscom
* Description : This short subroutine is uploaded to RAM by
*               the monitor program to perform ROM-writing
                tasks (because ROM is useless to the CPU
                during write operations)
*-----------------------------------------------------------
N_CONFIRMS  EQU 2 ; How many times should the toggle bit be verified?

            ORG     $17F000 ; Location at which this routine is moved into RAM
; ******** Calling parameters **********
; D6: Toggle bit status
; A0: RAM (Source) Start Address
; A1: ROM (Dest)   Start Address
; A2: RAM (Source) End Address
; begin program
COPYRR_MAIN
            BSR ERASE_SECT ; Erase initial sector
COPYRR_LOOP MOVE.B (A0)+, D1 ; Move data into position for copying
            BSR WRITE_BYTE   ; Write
            LEA 1(A1), A1 ; Increment A1
            ; TODO further erases

            CMP.L A0, A2    ; If start != end,
            BNE COPYRR_LOOP ; then continue
            RTS


* Subroutine ROM_PRES_B
* Keys the 2-cycle software control codes into both rom chips
ROM_PRES_B  MOVE.W #$AAAA, $00AAAA
            MOVE.W #$5555, $005554
            RTS

ERASE_SECT
            BSR ROM_PRES_B
            MOVE.W #$8080, $00AAAA ;Erase sector pt1
            BSR ROM_PRES_B
            MOVE.W #$3030, (A1) ;Erase sector pt2
            BSR CP_toggle   Wait until erased
            ;ADDI #1, A1 ; High-toggling byte   TODO might not need to toggle both chips
            ;BSR CP_toggle   Wait until erased
            ;SUBI #1, A1 ; Restore A1
            RTS

; Sub: Wait for toggle indicator bit to stop toggling on hi ROM
;  A6: the address to be checked
CP_toggle   MOVE.B #1, D4      Clear D4 (ROL status)
            MOVE.B (A1), D1    Read toggle bit to D1
CP_toggle2
            BTST #N_CONFIRMS, D4 Test bit for N confirms
            BNE CP_tog_RET     If that bit is high, confirmation succeeded 3 times
            MOVE.B D1, D2      Save previous toggle bit
            MOVE.B (A1), D1    Read new toggle bit to D1
            EOR.B D1, D2       EOR new byte onto old one
            BTST #6, D2        If EOR indicates the byts are different (toggle bit6 = 1)
            BNE CP_toggle2     If same (0), continue.  If different (1), start over
            ROL.B #1, D4       Rotate D4 bit (same)
            BRA CP_toggle2     Return
CP_tog_RET  RTS
 ;----- End Copy Routine -----;

* Subroutine WRITE_BYTE
* D1: Data Byte
* A1: Address for write
* D3: Reserved
WRITE_BYTE
            MOVE.L A1, D3
            BTST #0, D3
            BNE WRITE_ODD     If set (1), go to odd
```

```
            ;Even/High
WRITE_EVEN  BSR ROM_PRES_B
            MOVE.W #$A000, $00AAAA ;Erase High pt2;Write pt 1
            MOVE.B D1, (A1)  ; Write byte
            ;MOVE.B D1, LED ; debug
            BSR CP_toggle
            RTS

            ;Odd/Low
WRITE_ODD   BSR ROM_PRES_B
            MOVE.W #$00A0, $00AAAA ;Erase High pt2;Write pt 1
            MOVE.B D1, (A1)  ; Write byte
            ;MOVE.B D1, LED ; debug
            BSR CP_toggle
            RTS

            END COPYRR_MAIN
```

2. Monitor Program Code Listing

```
*-----------------------------------------------------------
* Title      : Monitor Program
* Written by : Chandler Griscom
*-----------------------------------------------------------

SIM         EQU 0       ;0 = hardware state, 1 = simulation state

BUFFER_A_SP EQU $104000 ;DUART circular buffer
BUFFER_A_EP EQU $104004
BUFFER_A_S  EQU $104008
BUFFER_A_E  EQU $104200

BUFFER_R_S  EQU $110004 ; Use to buffer data into RAM before copying to ROM
BUFFER_R_P  EQU $110000

COPYRR_MAIN EQU $17F000 ; Location of copy ROM routine

ROMBurn_ROM EQU $006000 ; Location of ROMBurner program
USR1_ROM    EQU $008000 ; Location of User Program 1

SUPER_STACK EQU $103F00
TIMER_SEC   EQU $103F04
TIMER_MS    EQU $103F08
DUINT_DISABLE EQU $103F10 ; Interrupts disabled? -> #$00
ECHO_MEM    EQU $103F11 ; Echo On -> #$FF
TIMER_MEM   EQU $103F12 ; Timer LEDs On -> #$FF


CPLD_STATUS EQU $300001

LED         EQU $F00001

MR1A        EQU $200001 Mode Register1
MR2A        EQU $200001 points here after MR1A is set
SRA         EQU $210001 Status Register (read)
CSRA        EQU $210001 Clock Select Register
CRA         EQU $220001 Command Register
TBA         EQU $230001 Transfer Holding Register
RBA         EQU $230001 Receive Holding Register
ACR         EQU $240001 Auxiliary control register

MR1B        EQU $280001 Mode Register1
MR2B        EQU $280001 points here after MR1A is set
SRB         EQU $290001 Status Register (read)
CSRB        EQU $290001 Clock Select Register
CRB         EQU $2A0001 Command Register
TBB         EQU $2B0001 Transfer Holding Register
RBB         EQU $2B0001 Receive Holding Register

IMR         EQU $250001 Interrupt Mask Register
ICR         EQU $250001 Interrupt control register
IVR         EQU $2C0001 Interrupt vector register

RxRDY       EQU 0       Recieve ready bit position
TxRDY       EQU 2       Transmit ready bit position
BAUD        EQU $CC     baud rate value = 19200 baud
```

```
CR          EQU $0D
LF          EQU $0A

START       ORG     $000000
            DC.L    SUPER_STACK Initial Stack Pointer
            DC.L    MAIN        Initial PC
            DC.L    EXCEPTION   Berr
            DC.L    EXCEPTION   Address Error
            DC.L    EXCEPTION   Illegal Instruction
            DC.L    EXCEPTION   Div by zero

            ORG     $000100   Interrupt vectors
            DC.L    $112000   64: Should be GPIO IRQ
            DC.L    IPL6      65: Periodic timer interrupt
            DC.L    DUART_IRQ 66: DUART RxRDYA or RXRDYB

     ; TIMER INTERRUPT HANDLER
            ORG     $000400
IPL6        MOVE.L  D0, -(SP)
            MOVE.W  TIMER_MS, D0
            ADD.W   #$1, D0
            MOVE.W  D0, TIMER_MS
            CMPI.W  #$03E7,D0
            BLS.S   IRQ6_QUIT
            CLR.W   TIMER_MS
            MOVE.L  TIMER_SEC, D0
            ADD.L   #$1, D0
            MOVE.L  D0, TIMER_SEC

            CMP.B #$FF, TIMER_MEM If timer off,
            BNE IRQ6_QUIT quit
            MOVE.B  D0, LED ; Else display the timer LED


IRQ6_QUIT   MOVE.L  (SP)+, D0
            RTE

; DUART Interrupt Handler

DUART_IRQ   OR.W  #$0700, SR ; Mask all interrupts
            MOVE.L D0,-(SP)
            MOVE.L A0,-(SP)
TestA

            MOVE.B SRA, D0  Read the A status register
            BTST #RxRDY, D0 Test reciever A ready status
            BEQ TestB  Goto B if no character in buffer
            MOVE.B RBA, D0  Else, Read the character into D0


            ; Circular queue write A
            MOVE.L BUFFER_A_EP, A0 Load the end pointer
            MOVE.B D0, (A0)+       Insert the read byte
            CMP.L #BUFFER_A_E, A0 If not at end of buffer,
            BNE BUFFER_A_WF       Branch to finish
            MOVE.L #BUFFER_A_S, A0 Move end pointer to beginning of buffer
BUFFER_A_WF MOVE.L A0, BUFFER_A_EP

            CMP.B #$FF, ECHO_MEM If echo off,
            BNE TestA Flush buffer
            JSr PUTCHAR_A Else putchar

            BRA TestA Flush buffer again (if nothing exists it will branch to TestB)
TestB

DU_Ret      MOVE.L (SP)+, A0
            MOVE.L (SP)+, D0
            AND.W #$F8FF, SR ; TODO this line is not necessary (?) because RTE pops SR
            RTE  Return from exception

EXCEPTION   LEA SUPER_STACK, SP ; Handle exception and reset
            MOVE.L #EXPSTRING, A0
            JSR PUTSTR_A
            JMP MAIN


            ; ***Begin program***
MAIN

            ; Init buffers
```

```
            MOVE.L #BUFFER_A_S, BUFFER_A_SP
            MOVE.L #BUFFER_A_S, BUFFER_A_EP

            ; ROM buffer is initialized by the ROM routine

            CLR.L  TIMER_MS
            CLR.L  TIMER_SEC

            JSR INIT_DUART

            MOVE.B #$00, ECHO_MEM ; Turn off echo A
            MOVE.B #$FF, TIMER_MEM ; Turn on timer LED


            JSR ENABLE_I ;Enable all interrupts

mLOOP       MOVE.L #MoniPrompt, A0 ; MAIN MENU
            JSR PUTSTR_A
mgetLOOP    JSR GETCHAR_A
            CMP.B #'a', D0 If user says 'a', administration menu
            BEQ aLOOP
            CMP.B #'c', D0 If user says 'c', view registers
            BEQ view_REG
            CMP.B #'C', D0 If user says 'C', edit registers
            BEQ edit_REG
            CMP.B #'m', D0 If user says 'm', view memory range
            BEQ view_MEM
            CMP.B #'M', D0 If user says 'm', edit memory
            BEQ edit_MEM
            CMP.B #'S', D0 If user says 'S', upload userprog 1
            BEQ upload_USR1
            CMP.B #'x', D0 If user says 'x', execute userprog 1
            BEQ exec_USR1
            CMP.B #'s', D0 If user says 's', load and execute s record in RAM
            BEQ S_REC_RAM
            CMP.B #'r', D0 If user says 'r', refresh
            BEQ mLOOP
            BRA mgetLOOP

aLOOP       MOVE.L #AdminPrompt, A0 ; ADMINISTRATION
            JSR PUTSTR_A
agetLOOP    JSR GETCHAR_A
            CMP.B #'A', D0 If user says 'A', main menu
            BEQ mLOOP
            CMP.B #'b', D0 If user says 'b', execute ROMBurner
            BEQ exec_ROMB
            CMP.B #'B', D0 If user says 'B', write ROMBurner
            BEQ upload_ROMB
            CMP.B #'i', D0 If user says 'i', disable interrupts
            BEQ sDISABLE_I
            CMP.B #'I', D0 If user says 'I', enable interrupts
            BEQ sENABLE_I
            CMP.B #'t', D0 If user says 't', disable timer
            BEQ sDISABLE_T
            CMP.B #'T', D0 If user says 'T', enable timer
            BEQ sENABLE_T
            CMP.B #'r', D0 If user says 'r', refresh
            BEQ aLOOP
            BRA agetLOOP

edit_REG    LEA RegPrompt, A0 ; Clear the screen
            JSR PUTSTR_A
            JSR GETCHAR_A ; Read register ID into D0
            JSR REG_sel
            JSR SCANLADD_A ; Read address into A0
            BCS viewMem_RET ; Error, return
ERegD0      CMP.B #'0', D0
            BNE ERegD1
            MOVE.L A0, D0
            BRA viewMem_RET ; Return
ERegD1      CMP.B #'1', D0
            BNE ERegD2
            MOVE.L A0, D1
            BRA viewMem_RET ; Return
ERegD2      CMP.B #'2', D0
            BNE ERegD3
            MOVE.L A0, D2
            BRA viewMem_RET ; Return
ERegD3      CMP.B #'3', D0
            BNE ERegD4
```

```
            MOVE.L A0, D3
            BRA viewMem_RET ; Return
ERegD4      CMP.B #'4', D0
            BNE ERegD5
            MOVE.L A0, D4
            BRA viewMem_RET ; Return
ERegD5      CMP.B #'5', D0
            BNE ERegD6
            MOVE.L A0, D5
            BRA viewMem_RET ; Return
ERegD6      CMP.B #'6', D0
            BNE ERegD7
            MOVE.L A0, D6
            BRA viewMem_RET ; Return
ERegD7      CMP.B #'7', D0
            BNE ERegA0
            MOVE.L A0, D7
            BRA viewMem_RET ; Return
ERegA0      CMP.B #')', D0
            BNE ERegA1
            MOVE.L A0, A0
            BRA viewMem_RET ; Return
ERegA1      CMP.B #'!', D0
            BNE ERegA2
            MOVE.L A0, A1
            BRA viewMem_RET ; Return
ERegA2      CMP.B #'@', D0
            BNE ERegA3
            MOVE.L A0, A2
            BRA viewMem_RET ; Return
ERegA3      CMP.B #'#', D0
            BNE ERegA4
            MOVE.L A0, A3
            BRA viewMem_RET ; Return
ERegA4      CMP.B #'$', D0
            BNE ERegA5
            MOVE.L A0, A4
            BRA viewMem_RET ; Return
ERegA5      CMP.B #'%', D0
            BNE ERegA6
            MOVE.L A0, A5
            BRA viewMem_RET ; Return
ERegA6      CMP.B #'^', D0
            BNE ERegA7
            MOVE.L A0, A6
            BRA viewMem_RET ; Return
ERegA7      CMP.B #'&', D0
            BNE ERegErr
            MOVE.L A0, A7
            BRA viewMem_RET ; Return
ERegErr     LEA RegError, A0
            JSR PUTSTR_A
            BRA viewMem_RET ; Error

            ; View Registers
view_REG    LEA RegPrompt, A0 ; Clear the screen
            JSR PUTSTR_A
            JSR GETCHAR_A ; Read register ID into D0
            JSR REG_sel
VRegD0      CMP.B #'0', D0
            BEQ VReg_Print
VRegD1      MOVE.L D1, A0
            CMP.B #'1', D0
            BEQ VReg_Print
VRegD2      MOVE.L D2, A0
            CMP.B #'2', D0
            BEQ VReg_Print
VRegD3      MOVE.L D3, A0
            CMP.B #'3', D0
            BEQ VReg_Print
VRegD4      MOVE.L D4, A0
            CMP.B #'4', D0
            BEQ VReg_Print
VRegD5      MOVE.L D5, A0
            CMP.B #'5', D0
            BEQ VReg_Print
VRegD6      MOVE.L D6, A0
            CMP.B #'6', D0
            BEQ VReg_Print
VRegD7      MOVE.L D7, A0
```

```
            CMP.B #'7', D0
            BEQ VReg_Print
VRegA0      MOVE.L A0, A0
            CMP.B #')', D0
            BEQ VReg_Print
VRegA1      MOVE.L A1, A0
            CMP.B #'!', D0
            BEQ VReg_Print
VRegA2      MOVE.L A2, A0
            CMP.B #'@', D0
            BEQ VReg_Print
VRegA3      MOVE.L A3, A0
            CMP.B #'#', D0
            BEQ VReg_Print
VRegA4      MOVE.L A4, A0
            CMP.B #'$', D0
            BEQ VReg_Print
VRegA5      MOVE.L A5, A0
            CMP.B #'%', D0
            BEQ VReg_Print
VRegA6      MOVE.L A6, A0
            CMP.B #'^', D0
            BEQ VReg_Print
VRegA7      MOVE.L A7, A0
            CMP.B #'&', D0
            BEQ VReg_Print
            LEA RegError, A0
            JSR PUTSTR_A
            BRA viewMem_RET ; Error
VReg_Print  JSR PRINTLADD_A
            BRA viewMem_RET ; Error

REG_sel     LEA D0s, A0
            CMP.B #'0', D0
            BEQ SReg_Print
            LEA D1s, A0
            CMP.B #'1', D0
            BEQ SReg_Print
            LEA D2s, A0
            CMP.B #'2', D0
            BEQ SReg_Print
            LEA D3s, A0
            CMP.B #'3', D0
            BEQ SReg_Print
            LEA D4s, A0
            CMP.B #'4', D0
            BEQ SReg_Print
            LEA D5s, A0
            CMP.B #'5', D0
            BEQ SReg_Print
            LEA D6s, A0
            CMP.B #'6', D0
            BEQ SReg_Print
            LEA D7s, A0
            CMP.B #'7', D0
            BEQ SReg_Print
            LEA A0s, A0
            CMP.B #')', D0
            BEQ SReg_Print
            LEA A1s, A0
            CMP.B #'!', D0
            BEQ SReg_Print
            LEA A2s, A0
            CMP.B #'@', D0
            BEQ SReg_Print
            LEA A3s, A0
            CMP.B #'#', D0
            BEQ SReg_Print
            LEA A4s, A0
            CMP.B #'$', D0
            BEQ SReg_Print
            LEA A5s, A0
            CMP.B #'%', D0
            BEQ SReg_Print
            LEA A6s, A0
            CMP.B #'^', D0
            BEQ SReg_Print
            LEA A7s, A0
            CMP.B #'&', D0
            BEQ SReg_Print
```

```
SReg_Print  JSR PUTSTR_A
            RTS

edit_MEM    JSR SCANADDR_A
            BCS viewMem_RET ; Error, return
            MOVE.B #$FF, ECHO_MEM ; Echo on
            MOVE.L A0, A1 ; Preserve address
            MOVE.B (A1), D0
            LEA Contents, A0
            JSR PUTSTR_A
            JSR BIN2HEX
            ROR.W #8, D0
            JSR PUTCHAR_A
            ROL.W #8, D0
            JSR PUTCHAR_A
            LEA NewContents, A0
            JSR PUTSTR_A
            CLR.B D1
            JSR GETCHAR_A
            JSR HEX2BIN
            BCS viewMem_RET ; Error, return
            LSL.B #4, D0
            OR.B D0, D1
            JSR GETCHAR_A
            JSR HEX2BIN
            BCS viewMem_RET ; Error, return
            OR.B D0, D1
            MOVE.B D1, (A1)
            MOVE.B #$00, ECHO_MEM ; Echo off
            BRA viewMem_RET

view_MEM    JSR SCANADDR_A
            BCS viewMem_RET ; Error, return
            MOVE.L A0, D3 ; Preserve address
            ; If address is odd, dec 1
            BTST #0, D3 ; If bit0 is not set (odd)
            BEQ viewMem_Vld
            SUBI.B #1, D3 ; Subtract 1
viewMem_Vld LEA ClearScr, A0 ; Clear the screen
            JSR PUTSTR_A
            MOVE.L D3, A0 ; Fetch address
            MOVE.W #15, D1 ; D1 contains row size minus one
viewMem_LoR   MOVE.B #'$', D0 ; Row loop; begin with address
              JSR PUTCHAR_A
              JSR PRINTADDR_A ; Print address in A0
              MOVE.B #':', D0
              JSR PUTCHAR_A
              MOVE.W #7, D2 ; D2 contains column size minus one
viewMem_LoC     MOVE.B #' ', D0 ; Column loop; begin with space
                JSR PUTCHAR_A
                MOVE.W (A0)+, D0 ; Load data word
                JSR BIN2HEX ; Convert word to long hex
                ROL.L #8, D0
                JSR PUTCHAR_A
                ROL.L #8, D0
                JSR PUTCHAR_A
                MOVE.B #' ', D0 Separate second word with a space
                JSR PUTCHAR_A
                ROL.L #8, D0
                JSR PUTCHAR_A
                ROL.L #8, D0
                JSR PUTCHAR_A

                DBRA D2, viewMem_LoC ; Loop Col
              MOVE.B #CR, D0 ; Break line
              JSR PUTCHAR_A
              MOVE.B #LF, D0
              JSR PUTCHAR_A
              DBRA D1, viewMem_LoR ; Loop Row

viewMem_RET LEA PressKey, A0
            JSR PUTSTR_A
            JSR GETCHAR_A
            BRA mLOOP

upload_ROMB MOVE.L #ROMBurn_ROM, D0
            JSR DISABLE_I ; Disable interrupts for uploads
            JSR S_REC_Burn
            JSR ENABLE_I ;Reenable interrupts
            BRA aLOOP
```

```
exec_ROMB   MOVE.L #ROMBurn_ROM, D0
            JSR DISABLE_I ;Disable all interrupts
            JSR S_REC_Exec
            JSR ENABLE_I ;Reenable interrupts
            BRA aLOOP

upload_USR1 MOVE.L #USR1_ROM, D0
            JSR DISABLE_I ; Disable interrupts for uploads
            JSR S_REC_Burn
            JSR ENABLE_I ;Reenable interrupts
            BRA mLOOP

exec_USR1   MOVE.L #USR1_ROM, D0
            JSR S_REC_Exec
            BRA mLOOP

sDISABLE_I  JSR DISABLE_I
            BRA aLOOP
sENABLE_I   JSR ENABLE_I
            BRA aLOOP

sDISABLE_T  MOVE.B #$00, TIMER_MEM
            BRA aLOOP
sENABLE_T   MOVE.B #$FF, TIMER_MEM
            BRA aLOOP

DISABLE_I   OR.W  #$0700, SR
            JSR DS_DUART_IR
            RTS

ENABLE_I    AND.W #$F8FF, SR
            JSR EN_DUART_IR
            RTS

; Puts copy RAM to ROM routine into memory
INIT_COPYRR LEA COPYCODE_S, A0 ; Source
            LEA COPYRR_MAIN, A1 ; Dest
            LEA COPYCODE_E, A2 ; End pointer

* Subroutine COPY_RANGE
* Copies range A0 through A2 to A1 through (A2-A0)+A1.
* Must be word-aligned
COPY_RANGE  MOVE.W (A0)+, (A1)+ ; Copy and advance pointers
            CMP.L A0, A2 ; If start!=end
            BNE COPY_RANGE ; Continue copying
            RTS ; Else return

; ROM upload executor - copies parsed srec to memory
S_REC_Exec  MOVE.L  D0, A1 ; Copy ROM Address argument tp
SREx_Loop1  CLR.L D1
            MOVE.B (A1)+, D1 ; Grab address from the parsed record - 3b
            ROR.L #8, D1
            MOVE.B (A1)+, D1
            ROR.L #8, D1
            MOVE.B (A1)+, D1
            ROR.L #8, D1 ; TODO replace with swap
            ROR.L #8, D1
            MOVE.L D1, A0 ; Move retrieved address to A0
            MOVE.B (A1)+, D1 ; Grab size long
            ROR.L #8, D1
            MOVE.B (A1)+, D1
            ROR.L #8, D1
            MOVE.B (A1)+, D1
            ROR.L #8, D1
            MOVE.B (A1)+, D1
            ROR.L #8, D1
            BEQ SREx_Exit ; If size is zero, execution has finished; exit.
SREx_Loop2  MOVE.B (A1)+, (A0)+ ; Copy data
            SUB.L #1,D1 ; Decrement then
            BNE   SREx_Loop2 ; branch if size != 0 (Didn't use DBRA because it's word-sized)
            BRA SREx_Loop1 ; Finished size loop; continue to next piece
SREx_Exit   MOVE.B #$00, TIMER_MEM ; Turn off timer LED
            JSR (A0) ; Execute the program!!!
            MOVE.B #$FF, TIMER_MEM ; Turn on timer LED
            RTS

; Direct RAM upload subroutines
S_REC_RAM   LEA SR_RAM_S12, A2 ; Load A2 with the S1/S2 routine
            LEA SR_RAM_S89, A3 ; Load A3 with the S8/S9 routine
```

```
            JSR S_REC_UP
            JMP mLOOP ; Go back to loop

SR_RAM_S12  MOVE.B D1, (A1) ; Copy into memory
            RTS

SR_RAM_S89  MOVE.L D1, A1 Move to A1
            MOVE.L #EOSString, A0 -- End of stream; press e to begin execution!!!
            JSR PUTSTR_A
            JSR GETCHAR_A
            CMP.B #$65, D0 If user says 'e', load s record
            BNE SR_RAM_Res Otherwise start over :(
            JMP     (A1)
SR_RAM_Res  RTS

; ROM upload subroutine
; Args: D0 contains the beginning ROM address
S_REC_Burn  MOVE.L  D0, -(SP) ; Store ROM Address argument
            JSR INIT_COPYRR ; Load the copy RAM to ROM routine into memory
            LEA SR_Burn_S12, A2 ; Load A2 with the S1/S2 routine
            LEA SR_Burn_S89, A3 ; Load A3 with the S8/S9 routine
            MOVE.L #$FFFFFFFF, D3 ; Use D3 for continuity checks! This register is not used
elsewhere
            MOVE.L #BUFFER_R_S, BUFFER_R_P ;ROM Buffer
            CLR.L D4 ; Clear size storage
            JSR S_REC_UP ; Upload S Record
            ; Blank out final null size long
            MOVE.B #0, (A4)
            MOVE.B #0, 1(A4)
            MOVE.B #0, 2(A4)
            MOVE.B #0, 3(A4)

            CMP.B #$FF, D0
            BEQ SRecBurnErr ; Error code; don't burn the rom!!

            LEA BUFFER_R_S, A0 ; Source starts at buffer start
            MOVE.L  (SP)+, A1 ; Destination: Pop ROM Address argument off the stack
            MOVE.L BUFFER_R_P, A2 ; Source ends at buffer pointer's final location (postincremented)
            JSR COPYRR_MAIN ; Copy ram to ROM!

SRecBurnErr MOVE.L #PressKey, A0 -- Press any key to continue
            JSR PUTSTR_A
            JSR GETCHAR_A

            RTS ; Go back to loop

            ; RAM SRec Buffer
            ; ADDRESS_3B - SIZE_4B - DATA_SizeBytes - ...(repeat)... - ADDRESS_3B (exec) - Null_5B
            ;D3 is free for use -- store continuity address
            ;D4 is free for use -- store size
            ;A4 is free for use -- store temporary size long pointer

            ;D1 contains a long address
SR_Burn_S89 MOVE.L D1, A1 ;Move to A1
            MOVE.L #BurnEnd, A0 ; Notify end of stream
            JSR PUTSTR_A
            MOVE.B #0, D1 ; Put a null byte into d1
            ; GOTO S1/2 routine and hope it works

            ;D6 Contains Record size (excluding checksum)
            ;D5 contains loop index (will equal address size on first iteration)
            ;D1 contains a data byte
            ;A1 contains the target address; use continuity checks
SR_Burn_S12 MOVE.L BUFFER_R_P, A0 ; Grab the buffer pointer
            CMP.L D3, A1
            BEQ SR_Burn_wrB ; Continuity check passed; write byte
            ; Else, write a new address
            CMP.L #0, D4 ; D4 is zero upon start
            BEQ SR_Burn_skS ; Skip writing size if we are just beginning

            MOVE.B D4, (A4) ; Store size long into reserved address from last time
            ROR.L #8, D4
            MOVE.B D4, 1(A4) ; Store size long into reserved address from last time
            ROR.L #8, D4
            MOVE.B D4, 2(A4) ; Store size long into reserved address from last time
            ROR.L #8, D4
            MOVE.B D4, 3(A4) ; Store size long into reserved address from last time
            ROR.L #8, D4
SR_Burn_skS
            MOVE.L A1, D3 ; Set continuity address
```

```
            ; Write word one byte at a time to avoid misalignment
            MOVE.B D3, (A0)+ ; Move current storage address into buffer and advance buffer pointer
            ROR.L #8, D3
            MOVE.B D3, (A0)+
            ROR.L #8, D3
            MOVE.B D3, (A0)+
            ROR.L #8, D3
            ROR.L #8, D3

            MOVE.L A0, A4    ; Store size long pointer so it can be written later
            LEA 4(A0), A0 ; Make space for data size byte
            CLR.B D4 ; Clear size

SR_Burn_wrB MOVE.B D1, (A0)+ ; Copy data byte into buffer
            ADD.L #1, D4 ; Increment size
            ADD.L #1, D3 ; Increment continuity address
            MOVE.L A0, BUFFER_R_P ; Store the buffer pointer
            RTS



 ;----- S Records -----;
 ; A2: Location of S1_2_Write
 ; A3: Location of S8_9_Write
S_REC_UP    BRA NEXTLN

            ; Eat checksum and carriage return
CRLF_NEXTLN JSR GETCHAR_A chksum
            JSR GETCHAR_A chksum
            JSR GETCHAR_A CR
            JSR GETCHAR_A LF

NEXTLN
            MOVE.L #SRecInsert, A0 -- Read next line
            JSR PUTSTR_A
            JSR GETCHAR_A S
            JSR GETCHAR_A Code
            MOVE.B D0, D7 Move code to D7

            CLR.L D6      Clear D6 for OR operations
            JSR GETCHAR_A Size MSB_Hex
            JSR HEX2BIN   Convert size to binary halfbyte
              BCS ERROR   ERROR HANDLING
            MOVE.B D0, D6 Copy to D6 for size record
            LSL.B #4, D6  Shift MSB left by 4
            JSR GETCHAR_A Size LSB_Hex
            JSR HEX2BIN   Convert size to binary halfbyte
              BCS ERROR   ERROR HANDLING
            OR.B D0, D6   OR it onto D6 for a complete size byte
            SUB.B #1, D6  Subtract 1 to ignore the checksum TODO add checksum functionality

            ; Clear working registers
            CLR.L D1
            CLR.L D5 ; Use d5 for loop index
SIZELOOP    CMP.W D6, D5 Break if i >= size
            BHS CRLF_NEXTLN
            ADD #$1, D5 i++
            ; READ AND CONVERT BYTE
            LSL.L #4, D1  Shift total left by 4
            JSR GETCHAR_A Addr MSB_Hex
            JSR HEX2BIN   Convert addr to binary halfbyte
              BCS ERROR   ERROR HANDLING
            OR.B D0, D1   Or onto D1 for addr record
            LSL.L #4, D1  Shift total left by 4
            JSR GETCHAR_A Addr LSB_Hex
            JSR HEX2BIN   Convert addr to binary halfbyte
              BCS ERROR   ERROR HANDLING
            OR.B D0, D1   Or onto D1 for addr record
            ; END READ AND CONVERT
S0          CMP.B #$30, D7 If this is a S0 record...
            BNE S1

            ; S0 Code: Informational Data (echo ASCII)
            MOVE.B D1, D0
            JSR PUTCHAR_A
            ; End S0 Code
            BRA SIZELOOP  Branch back to loop

S1          CMP.B #$31, D7 If this is a S1 record...
            BNE S2
```

```
            ; S1 Code: 16-bit address data
            CMP.W #1, D5  If i = 1 (incomplete)
            BEQ SIZELOOP  Continue

            CMP.W #2, D5  If i = 2 (address is ready for setting)
            BNE S1_2_Write  Write the data if i /= 3
            MOVE.L D1, A1 Copy resulting base address to A1
            CLR.L D1      Clear D1 for next round
            BRA SIZELOOP  Branch back to loop

S1_2_Write  ; Else this is i > 3
            JSR (A2) ; Jump to reserved location
            ADD #1, A1 Advance to next byte
            CLR.L D1      Clear D1 for next round
            ; End S1 Code
            BRA SIZELOOP Branch back to loop


S2          CMP.B #$32, D7 If this is a S2 record...
            BNE S8

            ; S2 Code: 24-bit address data
            CMP.W #1, D5  If i = 1 (incomplete)
            BEQ SIZELOOP  Continue

            CMP.W #2, D5  If i = 2 (incomplete)
            BEQ SIZELOOP  Continue

            CMP.W #3, D5  If i = 3 (address is ready for setting)
            BNE S1_2_Write  Write the data if i /= 3
            MOVE.L D1, A1 Copy resulting base address to A1
            CLR.L D1      Clear D1 for next round
            BRA SIZELOOP  Branch back to loop


S8          CMP.B #$38, D7 If this is a S8 record...
            BNE S9

            ; S8 Code: 24-bit address
S8_Valid    CMP.W D6, D5  Continue if i /= size
            BNE SIZELOOP Else execute record
            ; Eat carriage return and checksum
            JSR GETCHAR_A chk
            JSR GETCHAR_A chk
            JSR GETCHAR_A CR
            JSR GETCHAR_A LF

            JSR (A3) ; Jump to supplied execution routine.  D1 Contains valid address
            RTS ; If that sub decides to return, return again

S9          CMP.B #$39, D7 If this is a S9 record...
            BNE ERROR If it's not an S9 at this point something went wrong
            BRA S8_Valid S8 code does the same thing

            ; Else Error!
ERROR       MOVE.L #SRecError, A0
            JSR PUTSTR_A
            MOVE.B #$FF, D0 ; Indicate error
            RTS ; Return

 ;------ S Records ------;


**************** SHARED SUBROUTINES ******************
        ORG $001000


* Subroutine DEC2BIN -- Convert ASCII Decimal to Binary
* Inputs: D0 - ASCII Byte
* Output: D0 - Converted Binary Byte
*         Carry Bit - set to 1 on error
DEC2BIN:    CMP.B   #$30, D0   ; Check if less than ASCII 0
            BLO     DEC2BINERR
            CMP.B   #$39, D0   ; Check if greater than ASCII 9
            BHI     DEC2BINERR
            SUB.B   #$30, D0   ; Subtract 30 (ASCII 0) and move on
            RTS
DEC2BINERR: ORI     #$01, SR   ; Error, set Status Register carry bit
            RTS
```

```
* Subroutine HEX2BIN -- Convert ASCII Hex to Binary
* Inputs: D0 - ASCII Byte
* Output: D0 - Converted Binary Byte
*         Carry Bit - set to 1 on error
HEX2BIN:    CMP.B   #$30, D0
            BLO     HEX2BINERR ; Less than ASCII 0; error
            CMP.B   #$39, D0
            BHI     HEX2BIN_UC ; Greater than ASCII 9; branch to uppercase check
            SUB.B   #$30, D0   ; Within decimal range; subtract and return
            RTS
HEX2BIN_UC: CMP.B   #$41, D0
            BLO     HEX2BINERR ; Less than ASCII A; error
            CMP.B   #$46, D0
            BHI     HEX2BIN_LC ; Greater than ASCII F; branch to lowercase check
            SUB.B   #$37, D0   ; Within uppercase range; subtract and return
            RTS
HEX2BIN_LC: CMP.B   #$61, D0
            BLO     HEX2BINERR ; Less than ASCII a; error
            CMP.B   #$66, D0
            BHI     HEX2BINERR ; Greater than ASCII f; branch to lowercase check
            SUB.B   #$57, D0   ; Within lowercase range; subtract and return
            RTS
HEX2BINERR: ORI     #$01, SR   ; Error, set Status Register carry bit
            RTS

* Subroutine BIN2HEX -- Convert Binary to ASCII Hexadecimal
* Inputs: D0 - Binary Word
* Output: D0 - 4-byte ASCII String
*         Carry Bit - set to 1 on error
BIN2HEX:    MOVE.W  D1, -(SP)      ;D1 is used for rotate operations; push the old one onto stack
            MOVE.W  D0, D1         ;Move D0 to D1 for rotate operations; D0 will contain final result

            AND.B   #$0F, D0       ;Mask out the left 4 bits
            ADD.B   #$30, D0       ;Adjust for ASCII
            CMP.B   #$3A, D0       ;Compare with ASCII '9' + 1
            BLO     B2H_Byte2      ;If within range, go on
            ADD.B   #07, D0        ;Add 7 more to bring into the uppercase range

B2H_Byte2:  ROR.W   #4, D1         ;Rotate source to next half-byte
            ROR.L   #8, D0         ;Rotate destination to the next byte
            MOVE.B  D1, D0         ;Copy it over
            AND.B   #$0F, D0       ;Mask out the left 4 bits
            ADD.B   #$30, D0       ;Adjust for ASCII
            CMP.B   #$3A, D0       ;Compare with ASCII '9' + 1
            BLO     B2H_Byte3      ;If within range, go on
            ADD.B   #07, D0        ;Add 7 more to bring into the uppercase range

B2H_Byte3:  ROR.W   #4, D1         ;Rotate source to next half-byte
            ROR.L   #8, D0         ;Rotate destination to the next byte
            MOVE.B  D1, D0         ;Copy it over
            AND.B   #$0F, D0       ;Mask out the left 4 bits
            ADD.B   #$30, D0       ;Adjust for ASCII
            CMP.B   #$3A, D0       ;Compare with ASCII '9' + 1
            BLO     B2H_Byte4      ;If within range, go on
            ADD.B   #07, D0        ;Add 7 more to bring into the uppercase range

B2H_Byte4:  ROR.W   #4, D1         ;Rotate source to next half-byte
            ROR.L   #8, D0         ;Rotate destination to the next byte
            MOVE.B  D1, D0         ;Copy it over
            AND.B   #$0F, D0       ;Mask out the left 4 bits
            ADD.B   #$30, D0       ;Adjust for ASCII
            CMP.B   #$3A, D0       ;Compare with ASCII '9' + 1
            BLO     B2H_End        ;If within range, finish
            ADD.B   #07, D0        ;Add 7 more to bring into the uppercase range

B2H_End:    ROR.L   #8, D0         ;One more rotate so that D0 is in correct order
            MOVE.W  (SP)+, D1      ;Restore D1
            RTS

; GETCHAR_A reads a character from DUART A into D0
GETCHAR_A IF.B SIM <EQ> #00 THEN.L  --Hardware Code--
            MOVE.L A0,-(SP)
            MOVE.L D1, -(SP)
In_buff_A   MOVE.L BUFFER_A_SP, D1
            CMP.L BUFFER_A_EP, D1
            BNE READ_BUFFA Start and end pointer are not equal; read buffer character

            ; Otherwise poll the DUART
            CMP.B #$00, DUINT_DISABLE
            BNE In_buff_A ; Interrupts are enabled; don't poll
```

```
In_poll_A   MOVE.B SRA, D1  Read the A status register
            BTST #RxRDY, D1 Test reciever ready status
            BEQ In_poll_A   UNTIL char recieved
            MOVE.B RBA, D0  Read the character into D0
            JMP READ_RETA

            ; Circular queue read
READ_BUFFA  MOVE.L BUFFER_A_SP, A0 Load the start pointer
            MOVE.B (A0)+, D0       Extract the read byte
            CMP.L #BUFFER_A_E, A0 If not at end of buffer,
            BNE BUFFER_A_RF       Branch to finish
            MOVE.L #BUFFER_A_S, A0
BUFFER_A_RF MOVE.L A0, BUFFER_A_SP
READ_RETA   MOVE.L (SP)+, D1
            MOVE.L (SP)+, A0
         ELSE                    --Simulation code--
            MOVE.L D1, -(SP)
            MOVE.L #05, D0
            TRAP   #15
            MOVE.B D1, D0
            MOVE.L (SP)+, D1
         ENDI
            RTS

; PUTCHAR_A outputs D0 to the DUART channel A
PUTCHAR_A IF.B SIM <EQ> #00 THEN.L  --Hardware Code--
            MOVE.L D1, -(SP)
Out_poll_A  MOVE.B SRA, D1
            BTST   #TxRDY, D1
            BEQ    Out_poll_A
            MOVE.B D0, TBA
            MOVE.L (SP)+, D1
          ELSE                     --Simulation code--
            MOVE.L D0, -(SP) ; Task
            MOVE.L D1, -(SP) ;Char to display
            MOVE.B D0, D1
            MOVE.L #06, D0
            TRAP   #15
            MOVE.L (SP)+, D1
            MOVE.L (SP)+, D0
          ENDI
            RTS


            ; Print String in A0
PUTSTR_A    MOVE.W D0, -(SP)
            MOVE.W D1, -(SP)
            CLR.W D1 Length will be stored here
pst_Next    CMP.W #$200, D1 If >= 512,
            BHS pst_Quit   quit
            MOVE.B (A0)+, D0
            ADD #1, D1
            CMP.B #0, D0   If null,
            BEQ pst_Quit   quit
            JSR PUTCHAR_A
            BRA pst_Next
pst_Quit    MOVE.W (SP)+, D1
            MOVE.W (SP)+, D0
            RTS


            ; GETCHAR_B reads a character from DUART B into D0
GETCHAR_B   MOVE.B SRB, D0  Read the A status register
            BTST #RxRDY, D0 Test reciever ready status
            BEQ GETCHAR_B   UNTIL char recieved
            MOVE.B RBB, D0  Read the character into D0
            RTS


            ; PUTCHAR_B outputs D0 to the DUART channel B
PUTCHAR_B   MOVE.L D1, -(SP)
Out_poll_B  MOVE.B SRB, D1
            BTST   #TxRDY, D1
            BEQ    Out_poll_B
            MOVE.B D0, TBB
            MOVE.L (SP)+, D1
            RTS
```

```
* Subroutine SCANADDR_A *
* Reads a 3-byte memory address from the user into A0
* Sets carry bit on error
SCANADDR_A  MOVE.W D0, -(SP)
            MOVE.L D1, -(SP)
            MOVE.W D2, -(SP)
            LEA MemPrompt, A0
            JSR PUTSTR_A
            MOVE.B #$FF, ECHO_MEM ; Turn on echo A
            MOVE.W #5, D2 ; Set loop to execute 6 times
            CLR.L D1 ; Clear D1, the address storage
ScnAdd_LOOP JSR GETCHAR_A ; Get a character
            JSR HEX2BIN ; Convert to a value between 0 and 15
            BCS ScnAdd_ERR ; Error report
            LSL.L #4, D1 ; Shift D1 left 4 bits
            OR.B D0, D1 ; Or the 4-bit value onto D1
            DBRA D2, ScnAdd_LOOP ; Repeat
            MOVE.B #$00, ECHO_MEM ; Turn off echo A
            LEA CRLF, A0 ; Break string
            JSR PUTSTR_A
            MOVE.L D1, A0 ; Copy result into A0
            MOVE.W (SP)+, D2 ; Restore stack
            MOVE.L (SP)+, D1
            MOVE.W (SP)+, D0
            RTS
ScnAdd_ERR  LEA MemError, A0
            JSR PUTSTR_A
            MOVE.W (SP)+, D2 ; Restore stack
            MOVE.L (SP)+, D1
            MOVE.W (SP)+, D0
            ORI #$01, SR ; Set carry bit
            RTS


* Subroutine SCANLADD_A *
* Reads 4 bytes into A0
* Sets carry bit on error
SCANLADD_A  MOVE.W D0, -(SP)
            MOVE.L D1, -(SP)
            MOVE.W D2, -(SP)
            LEA Write4b, A0
            JSR PUTSTR_A
            MOVE.B #$FF, ECHO_MEM ; Turn on echo A
            MOVE.W #7, D2 ; Set loop to execute 8 times
            CLR.L D1 ; Clear D1, the address storage
ScnLad_LOOP JSR GETCHAR_A ; Get a character
            JSR HEX2BIN ; Convert to a value between 0 and 15
            BCS ScnLad_ERR ; Error report
            LSL.L #4, D1 ; Shift D1 left 4 bits
            OR.B D0, D1 ; Or the 4-bit value onto D1
            DBRA D2, ScnLad_LOOP ; Repeat
            MOVE.B #$00, ECHO_MEM ; Turn off echo A
            LEA CRLF, A0 ; Break string
            JSR PUTSTR_A
            MOVE.L D1, A0 ; Copy result into A0
            MOVE.W (SP)+, D2 ; Restore stack
            MOVE.L (SP)+, D1
            MOVE.W (SP)+, D0
            RTS
ScnLad_ERR  LEA MemError, A0
            JSR PUTSTR_A
            MOVE.W (SP)+, D2 ; Restore stack
            MOVE.L (SP)+, D1
            MOVE.W (SP)+, D0
            ORI #$01, SR ; Set carry bit
            RTS

* Subroutine PRINTADDR_A (print a three-byte address) *
* Prints the memory address in A0
PRINTADDR_A MOVE.W D0, -(SP)
            MOVE.L D1, -(SP)
            MOVE.L A0, D1 ; Copy
            SWAP D1 ; Bring the upper word to the front $xx123456 -> $3456xx12
            MOVE.W D1, D0
            JSR BIN2HEX
            ROR.L #8, D0 ; Grab the upper part of the hex code
            JSR PUTCHAR_A
            ROL.L #8, D0 ; Revert to lower hex char
            JSR PUTCHAR_A
```

```
            SWAP D1 ; Swap over to the LSB
            MOVE.W D1, D0
            JSR BIN2HEX
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A

            MOVE.L (SP)+, D1
            MOVE.W (SP)+, D0
            RTS

* Subroutine PRINTLADD_A (print a 4-byte address) *
* Prints the entire A0
PRINTLADD_A MOVE.W D0, -(SP)
            MOVE.L D1, -(SP)
            MOVE.L A0, D1 ; Copy
            SWAP D1 ; Bring the upper word to the front $xx123456 -> $3456xx12
            MOVE.W D1, D0
            JSR BIN2HEX
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A

            SWAP D1 ; Swap over to the LSB
            MOVE.W D1, D0
            JSR BIN2HEX
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A
            ROL.L #8, D0
            JSR PUTCHAR_A

            MOVE.L (SP)+, D1
            MOVE.W (SP)+, D0
            RTS


INIT_DUART  ;Reset Duart
            MOVE.B #30, CRA
            MOVE.B #20, CRA
            MOVE.B #10, CRA
            MOVE.B #$B0,ACR    selects baud rate set 2, counter mode div by 16

            MOVE.B #BAUD,CSRA  set 19.2k (1: 36.4k) baud Rx/Tx
            MOVE.B #$13,MR1A   8-bits, no parity, 1 stop bit
            MOVE.B #$07,MR2A   07 sets: Normal mode, CTS and RTS disabled, stop bit length = 1
            MOVE.B #$05,CRA    enable Tx and Rx

            MOVE.B #BAUD,CSRB  set 19.2k (1: 36.4k) baud Rx/Tx
            MOVE.B #$13,MR1B   8-bits, no parity, 1 stop bit
            MOVE.B #$07,MR2B   07 sets: Normal mode, CTS and RTS disabled, stop bit length = 1
            MOVE.B #$05,CRB    enable Tx and Rx

            MOVE.B #66, IVR    set interrupt vector - dec 66
            JSR EN_DUART_IR    enable DUART interrupts

            RTS

            ;Enable Duart interrupts
EN_DUART_IR MOVE.B #$22, IMR    set interrupt masks to just RxRDYA, RxRDYB
            MOVE.B #$FF, DUINT_DISABLE
            RTS

            ;Disable Duart interrupts
DS_DUART_IR MOVE.B #$00, IMR    set interrupt masks to nothing
            CLR.B DUINT_DISABLE
            RTS
```

```
                ;Begin with the terminal CLR sequence
MoniPrompt  DC.B $1B,'[2J',$1B,'[H','----------- MONITOR PROGRAM -----------',CR,LF
            DC.B                    'Press a for administrative tasks.',CR,LF
            DC.B                    'Press c to view a CPU register.',CR,LF
            DC.B                    'Press C to edit a CPU register.',CR,LF
            DC.B                    'Press m to view a memory address.',CR,LF
            DC.B                    'Press M to edit a memory address.',CR,LF
            DC.B                    'Press S to upload an S record.',CR,LF
            DC.B                    'Press x to execute the S record.',CR,LF
            DC.B                    'Press r to refresh.',CR,LF
            DC.B                    '----------------------------------------',CR,LF,0
AdminPrompt DC.B $1B,'[2J',$1B,'[H','----------- ADMINISTRATION -----------',CR,LF
            DC.B                    'Press A to return to main menu.',CR,LF
            DC.B                    'Press b to run ROMBurner.',CR,LF
            DC.B                    'Press B to upload ROMBurner S Record.',CR,LF
            DC.B                    'Press i to disable interrupts.',CR,LF
            DC.B                    'Press I to enable interrupts.',CR,LF
            DC.B                    'Press t to disable timer.',CR,LF
            DC.B                    'Press T to enable timer.',CR,LF
            DC.B                    'Press r to refresh.',CR,LF
            DC.B                    '----------------------------------------',CR,LF,0
RegPrompt   DC.B $1B,'[2J',$1B,'[H','---------- Select a Register ----------',CR,LF
            DC.B                    'Number:       D0 through D7',CR,LF
            DC.B                    'Shift+Number: A0 through A7',CR,LF
            DC.B                    '----------------------------------------',CR,LF,0
MemPrompt   DC.B $1B,'[2J',$1B,'[H','Enter a 6-digit hex memory address: $',0
MemError    DC.B CR,LF,'Error reading address.',CR,LF,0
RegError    DC.B CR,LF,'Invalid register code.',CR,LF,0
ClearScr    DC.B $1B,'[2J',$1B,'[H',0
RegSelect   DC.B                    'You selected: ',0
Contents    DC.B CR,LF,'Current Contents: ',0
NewContents DC.B CR,LF,'Enter a 2-digit hex value to write: ',0
Write4B     DC.B CR,LF,'Enter an 8-digit hex value to write: ',0
SRecInsert  DC.B '.',0
CRLF        DC.B CR,LF,0
SRecError   DC.B CR,LF,'Error reading S record.',CR,LF,0
EOSString   DC.B CR,LF,'End of stream; press e to transfer execution!',CR,LF,0
BurnEnd     DC.B CR,LF,'Reached end of S record.',CR,LF,0
EXPSTRING   DC.B CR,LF,'Encountered exception; resetting!',CR,LF,0
PressKey    DC.B CR,LF,'Press any key to continue...',CR,LF,0
D0s         DC.B 'Selected D0...',CR,LF,0
D1s         DC.B 'Selected D1...',CR,LF,0
D2s         DC.B 'Selected D2...',CR,LF,0
D3s         DC.B 'Selected D3...',CR,LF,0
D4s         DC.B 'Selected D4...',CR,LF,0
D5s         DC.B 'Selected D5...',CR,LF,0
D6s         DC.B 'Selected D6...',CR,LF,0
D7s         DC.B 'Selected D7...',CR,LF,0
A0s         DC.B 'Selected A0...',CR,LF,0
A1s         DC.B 'Selected A1...',CR,LF,0
A2s         DC.B 'Selected A2...',CR,LF,0
A3s         DC.B 'Selected A3...',CR,LF,0
A4s         DC.B 'Selected A4...',CR,LF,0
A5s         DC.B 'Selected A5...',CR,LF,0
A6s         DC.B 'Selected A6...',CR,LF,0
A7s         DC.B 'Selected A7...',CR,LF,0

; *** Copy RAM buffer to ROM subroutine machine code ***
            DC.W $0000 ; Dummy word alignment
COPYCODE_S  DC.B $61,$00,$00,$22,$12,$18,$61,$00,$00,$52,$43,$E9,$00,$01,$B5,$C8,$66,$F2,$4E,$75,$33,$FC,$AA,$AA
            DC.B $00,$00,$AA,$AA,$31,$FC,$55,$55,$55,$54,$4E,$75,$61,$EE,$33,$FC,$80,$80,$00,$00,$AA,$AA,$61,$E4
            DC.B $32,$BC,$30,$30,$61,$00,$00,$04,$4E,$75,$18,$3C,$00,$01,$12,$11,$08,$04,$00,$02,$66,$00,$00,$12
            DC.B $14,$01,$12,$11,$B3,$02,$08,$02,$00,$06,$66,$EC,$E3,$1C,$60,$E8,$4E,$75,$26,$09,$08,$03,$00,$00
            DC.B $66,$00,$00,$12,$61,$AE,$33,$FC,$A0,$00,$00,$00,$AA,$AA,$12,$81,$61,$C8,$4E,$75,$61,$9E,$33,$FC
            DC.B $00,$A0,$00,$00,$AA,$AA,$12,$81,$61,$B8,$4E,$75
COPYCODE_E  DC.W $0000 ; Dummy word alignment

            END     MAIN
```