

CECS 525-01

Fall 2011

Project 3

Copyright 2010, Eugene Rockey

Name	Report (20 Points)	Demo (10 Points)	Questions (10 Points)
Group 1 Nick Searcy: Luke Spicer Adrian Fletcher Conor Heine			

Motorola 68000 Lab 3 Hardware Expansion

[Group 1: Adrian Fletcher, Conor Heine, Nick Searcy, Luke Spicer]

CECS 525-01

Abstract

In this lab a port diagram of the Motorola MC68000 pin mapping was created for interfacing with the I/O expansion port of the minimal computer. The port connections were traced by utilizing the continuity test of an OHM meter. The firmware was then re-engineered so that the operating system (TS2MON) operates in supervisor mode, while user applications operate in user mode. In order to test the previous modes of operation as well as the modes of operation developed in this lab, a logic analyzer was utilized to capture the function control bits through the various mode scenarios. Two circuits were constructed for hardware expansion of the minimal computer board: 1) a reset circuit that adds push button reset ability and 2) a RAM expansion circuit. A user application was then developed to be executed in the expanded RAM space for testing. Finally the serial port configuration of the Renesas M16C/62P was determined in order to interface with a host PC via the RS232C port using HyperTerminal. A program was developed in order to transfer a message from the M16C/62P to HyperTerminal for confirmation.

Body

Part 1:

The continuity test of an OHM meter was utilized to trace each pin of the MC68000 to its respective I/O expansion port. The result of this mapping, the I/O expansion ports diagram, is shown below in Figure 1. Two expansion port pins were found to be Vcc (5VDC) and three were found to be GND.

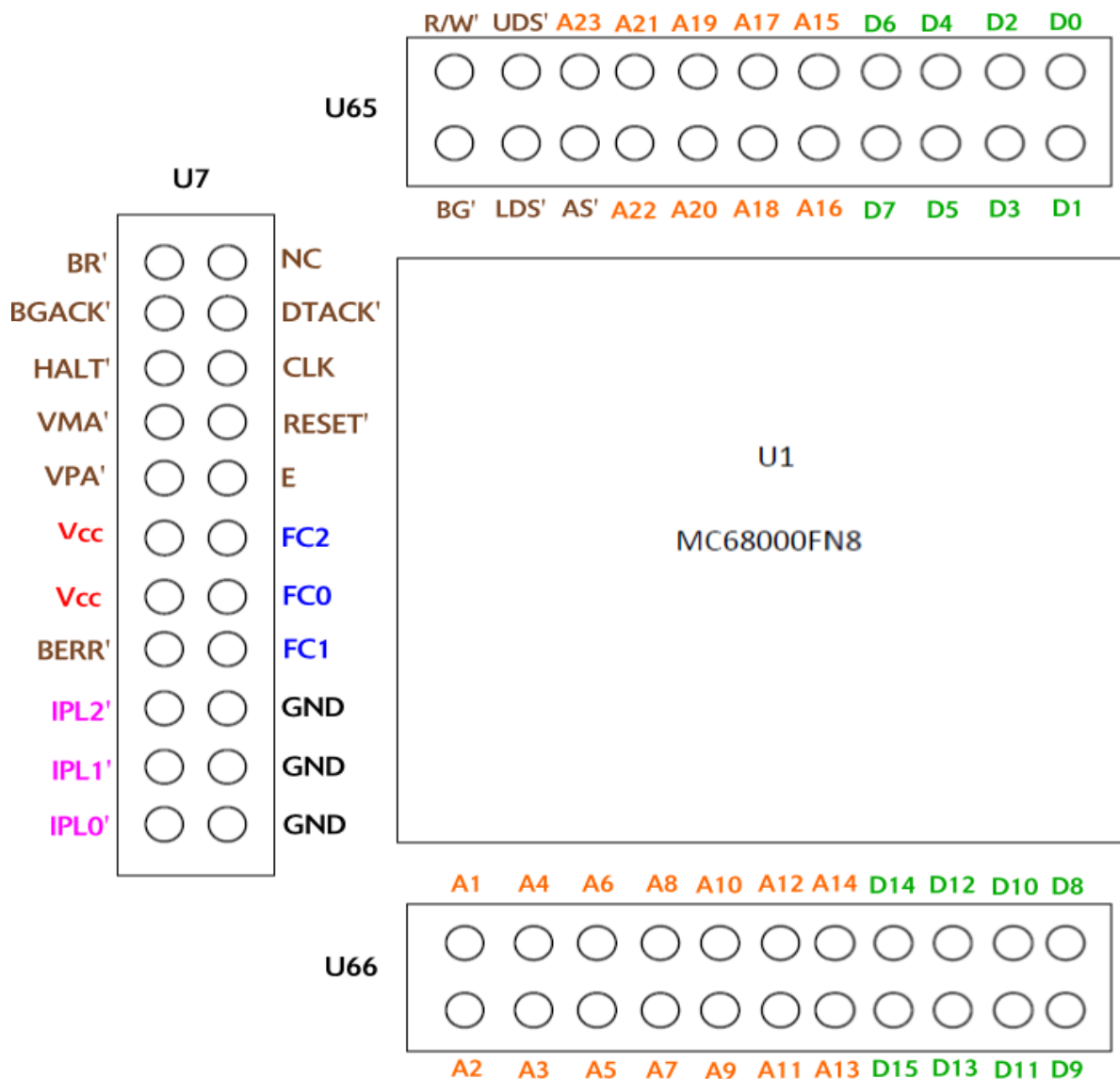


Figure 1. I/O Expansion Ports Diagram

Part 2:

The operating mode of the MC68000 were tested both before and after re-engineering the firmware. A logic analyzer was used to monitor the function control bits (FC2-FC0) under scenarios such as power up, running the OS, exception handling and running a user application. The operating modes were verified using Table 1 shown below.

Function Control (FC2:FC0)	Operating States
0	Reserved
1	User Data
2	User Program
3	Reserved
4	Reserved
5	Supervisor Data
6	Supervisor Program
7	Interrupt Acknowledge

Table 1. Function Control Bits to Operating States

The output of the function control bits for power-on-reset or reset scenarios are shown in Figure 2. The MC68000 alternates between the Supervisor Data (FC=5) and Reserved (FC=3) operating states, which indicates that the MC68000 is in supervisor mode upon power up or reset.

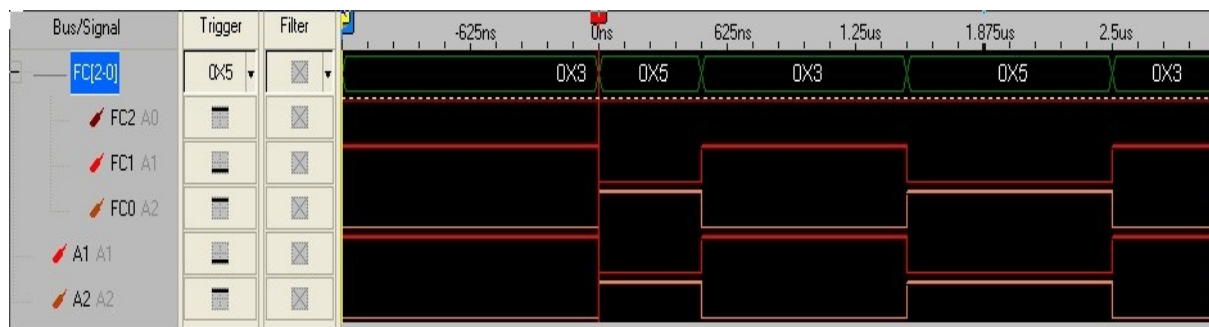


Figure 2. Operating States for Power-on-Reset and RESET prior to Re-Engineering the Firmware

The output of the function control bits during TS2MON execution are shown in Figure 3. The MC68000 continues to alternate between the supervisor data (FC=5) and Reserved (FC=3) states, which indicates that the MC68000 stays in supervisor mode.

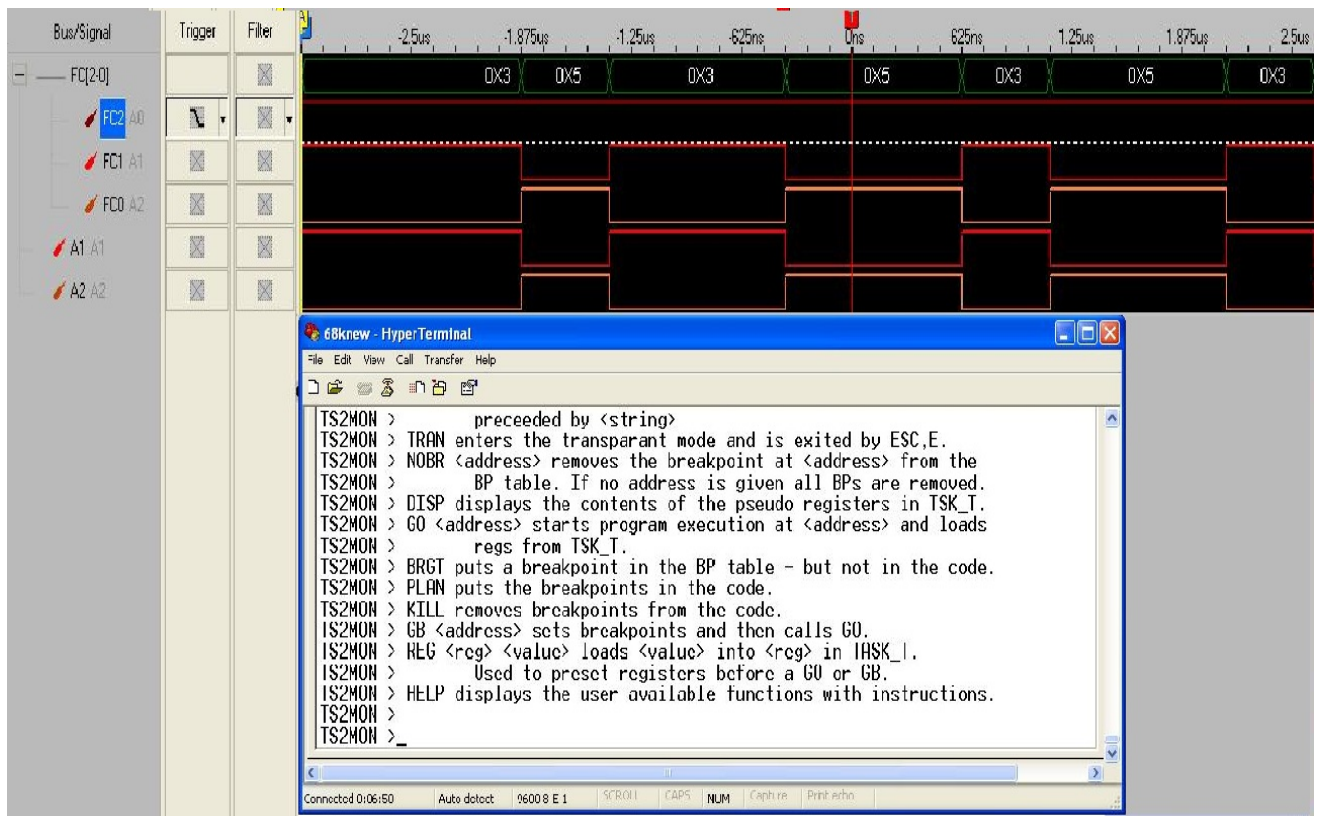


Figure 3. Operating States for TS2MON Execution Prior to Re-Engineering the Firmware

The output of the function control bits upon entering a user application are shown in Figure 4. The MC68000 continues to alternate between the supervisor data (FC=5) and Reserved (FC=3) states, which indicates that the MC68000 stays in supervisor mode.

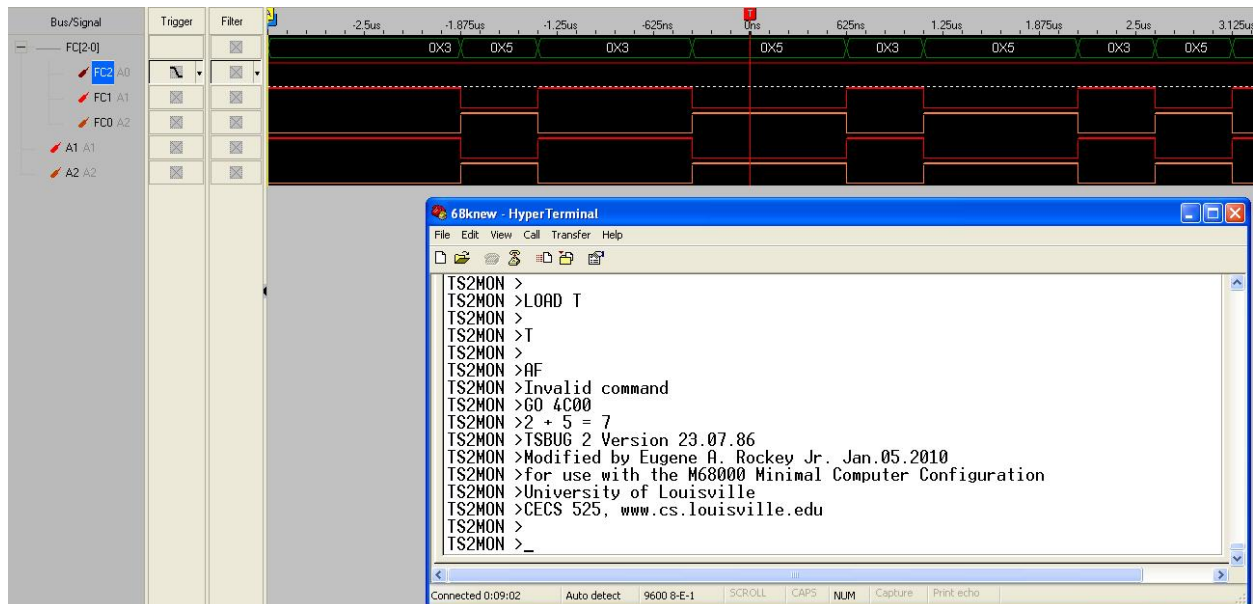


Figure 4. Operating States for User Program Execution Prior to Re-Engineering the Firmware

The contents of the status register while running the firmware on both hardware and the emulator are shown in Figure 5. In both cases, the status bit is set, indicating operation is in supervisor mode.

```

68knew - HyperTerminal
File Edit View Call Transfer Help

TS2MON >
TS2MON > Data reg      Address reg
TS2MON >0 00000004      00001E63
TS2MON >1 0000000F      00001E63
TS2MON >2 00000000      00004900
TS2MON >3 FFFFFFFF      00001BBA
TS2MON >4 FFFF7FFF      00003000
TS2MON >5 FFFFFFFF      000048C3
TS2MON >6 FFFFFFFF      00004800
TS2MON >7 00000000      FFFFFFFF
TS2MON >
TS2MON > SS = 000043E8
TS2MON > SR = 2704
TS2MON > PC = 00001BD6
TS2MON >
TS2MON >

Connected 0:01:45   Auto detect   9600 8-E-1   SCROLL   CAPS   NUM   Capture   Print echo

T S INT XNZVC Cycles
R=0010000000000000 38780
S=00FF0000 Clear Cycles
S=00000004 PC=00001700

TS2MON >TSBUG 2 Version 23.07.86
TS2MON >Modified by Eugene A. Rockey Jr. Jan.05.2010
TS2MON >For use with the M68000 Minimal Computer Configuration
TS2MON >University of Louisville
TS2MON >CECS 525, www.cs.louisville.edu
TS2MON >
TS2MON >

```

Figure 5. Status Register Contents During TS2MON Execution Prior to Re-Engineering the Firmware

Before implementing the “GO” fix, standard operation occurred in supervisor mode, even when running a user program. When an exception occurred in a user program, control remained in supervisor mode during and after the exception. This is shown by the function control codes and contents of the status register in Figure 6.

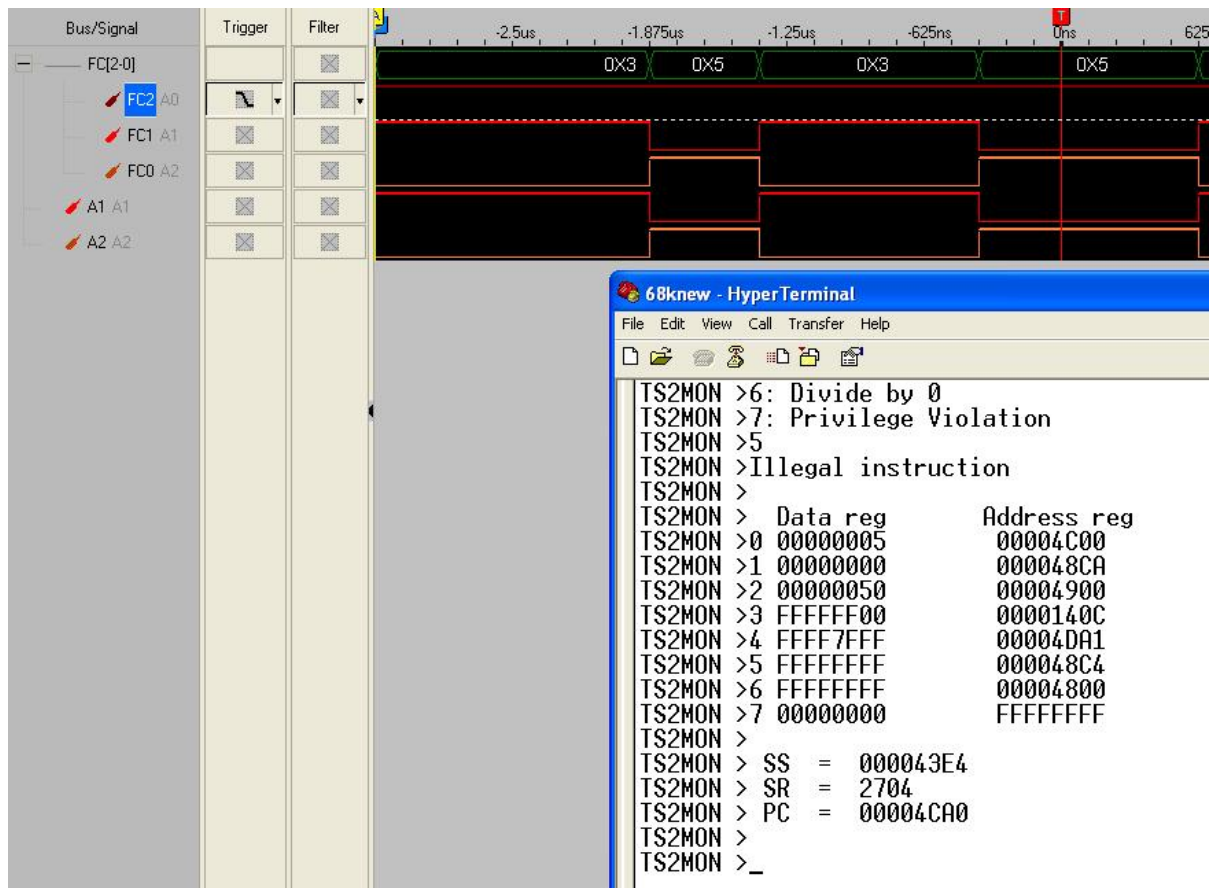


Figure 6. Operating States during Exception within a User Program Prior to Re-Engineering the Firmware

The results of the six operating mode proofs prior to re-engineering the firmware are shown below in Table 2.

Proof	Conclusion
Reset in Supervisor Mode	True
TS2MON Execution in Supervisor Mode	True
User application in Supervisor Mode	True
Interrupt in Supervisor Mode	True
Return from interrupt in Supervisor Mode	True
Return from user application in Supervisor Mode	True

Table 2. Operating Mode Proofs Prior to Re-Engineering Firmware

After verifying the undesired operating procedure, the “GO” routine was analyzed for

modifications necessary to provide the desired operation...

After re-engineering the firmware, the 68k is set to operate in supervisor mode upon reset. This is demonstrated by the function codes in Figure 7 and the status register contents in Figure 8 below.

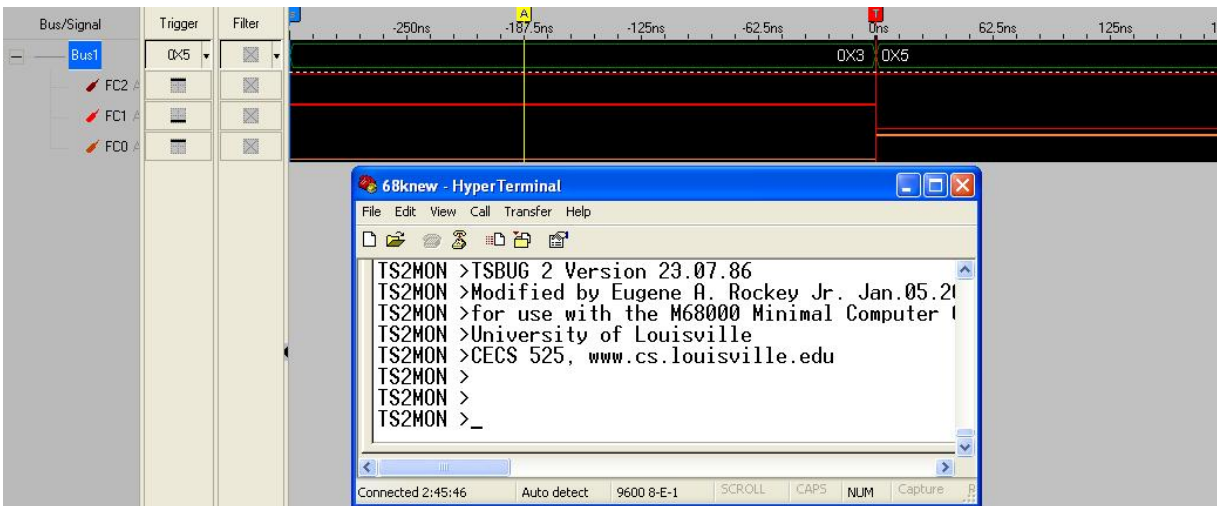
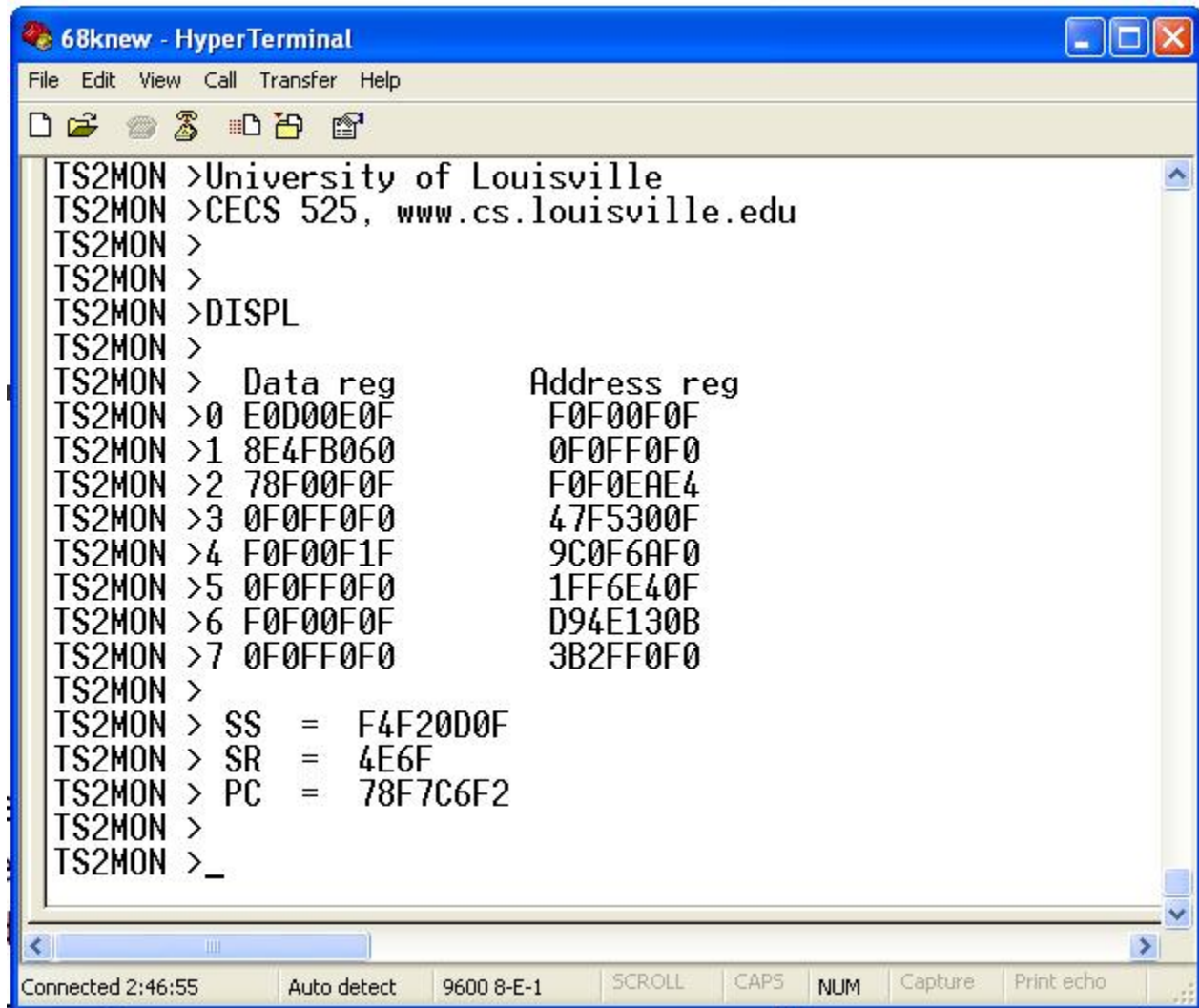


Figure 7. Operating States upon Reset After Re-Engineering Firmware



```
TS2MON >University of Louisville
TS2MON >CECS 525, www.cs.louisville.edu
TS2MON >
TS2MON >
TS2MON >DISPL
TS2MON >
TS2MON > Data reg      Address reg
TS2MON >0 E0D00E0F      F0F00F0F
TS2MON >1 8E4FB060      0F0FF0F0
TS2MON >2 78F00F0F      F0F0EAE4
TS2MON >3 0F0FF0F0      47F5300F
TS2MON >4 F0F00F1F      9C0F6AF0
TS2MON >5 0F0FF0F0      1FF6E40F
TS2MON >6 F0F00F0F      D94E130B
TS2MON >7 0F0FF0F0      3B2FF0F0
TS2MON >
TS2MON > SS = F4F20D0F
TS2MON > SR = 4E6F
TS2MON > PC = 78F7C6F2
TS2MON >
TS2MON >_
```

Connected 2:46:55 Auto detect 9600 8-E-1 SCROLL CAPS NUM Capture Print echo

Figure 8. Operating Status Register upon Reset After Re-Engineering Firmware

During TS2MON execution, the 68k remains in supervisor mode. When a user program is loaded and executed with the modified “GO” command, the status bit is set to user mode right before the executing the program. Once the user program has started, the 68k is operating in user mode. This is proven with the function codes displayed in Figure 9.

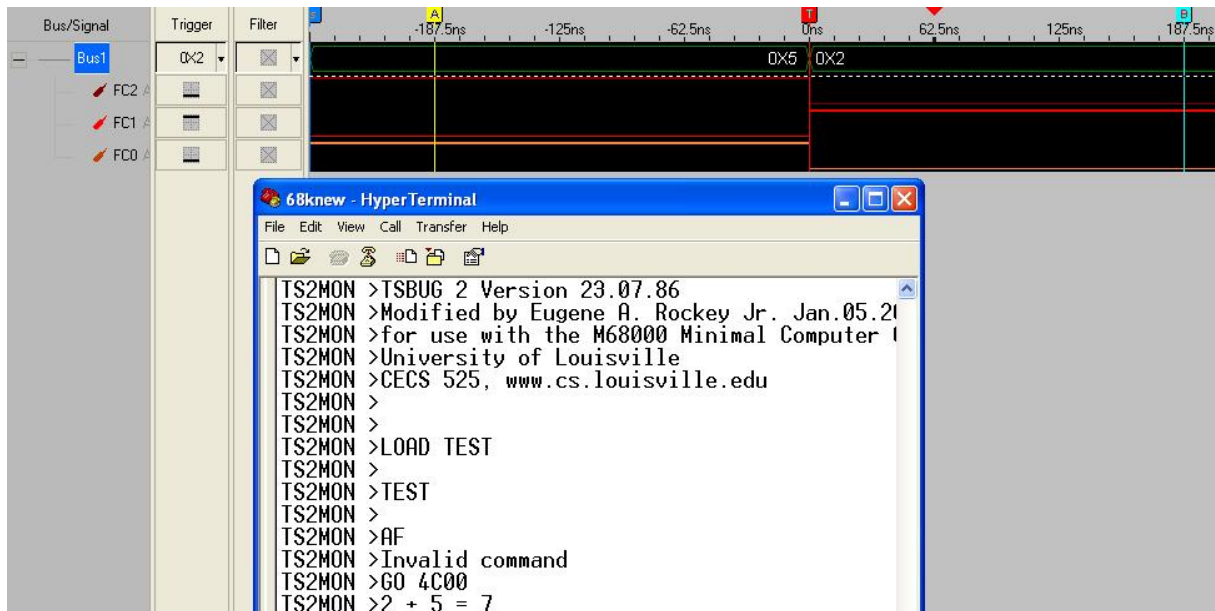


Figure 9. Operating States Immediately after Executing User Program After Re-Engineering Firmware

If an exception occurs at any point during execution of a user program either inadvertently or by request, then control is returned to the supervisor for the execution of the exception (as proven in Figure 10), and returned to user mode after the exception has completed (as proven in Figure 11).

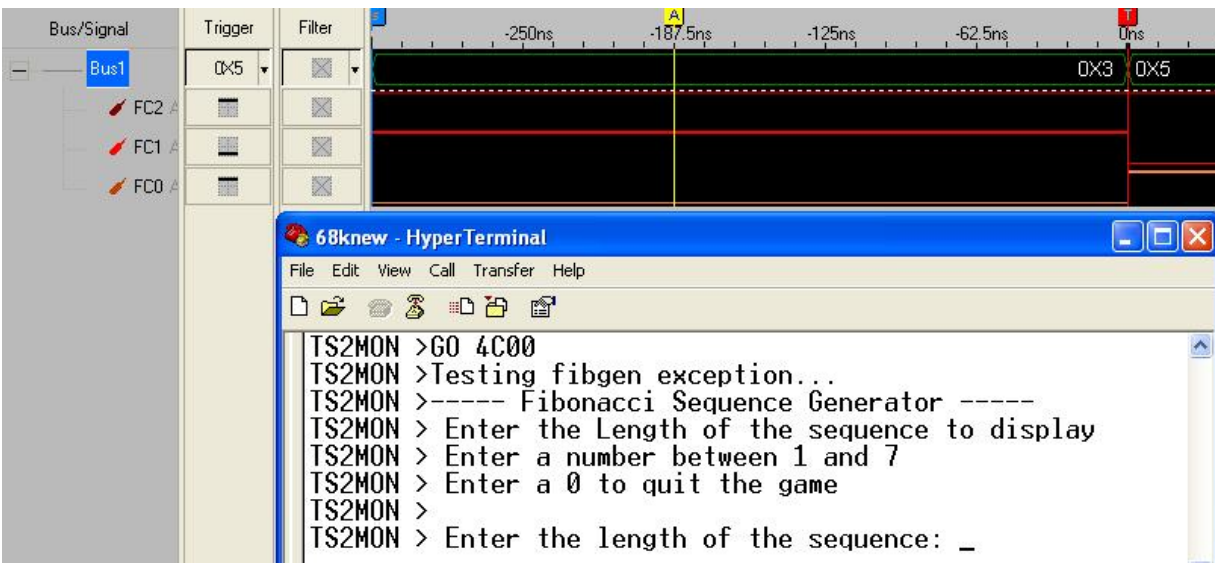


Figure 10. Operating States After Entering an Exception from a User Program After Re-Engineering Firmware

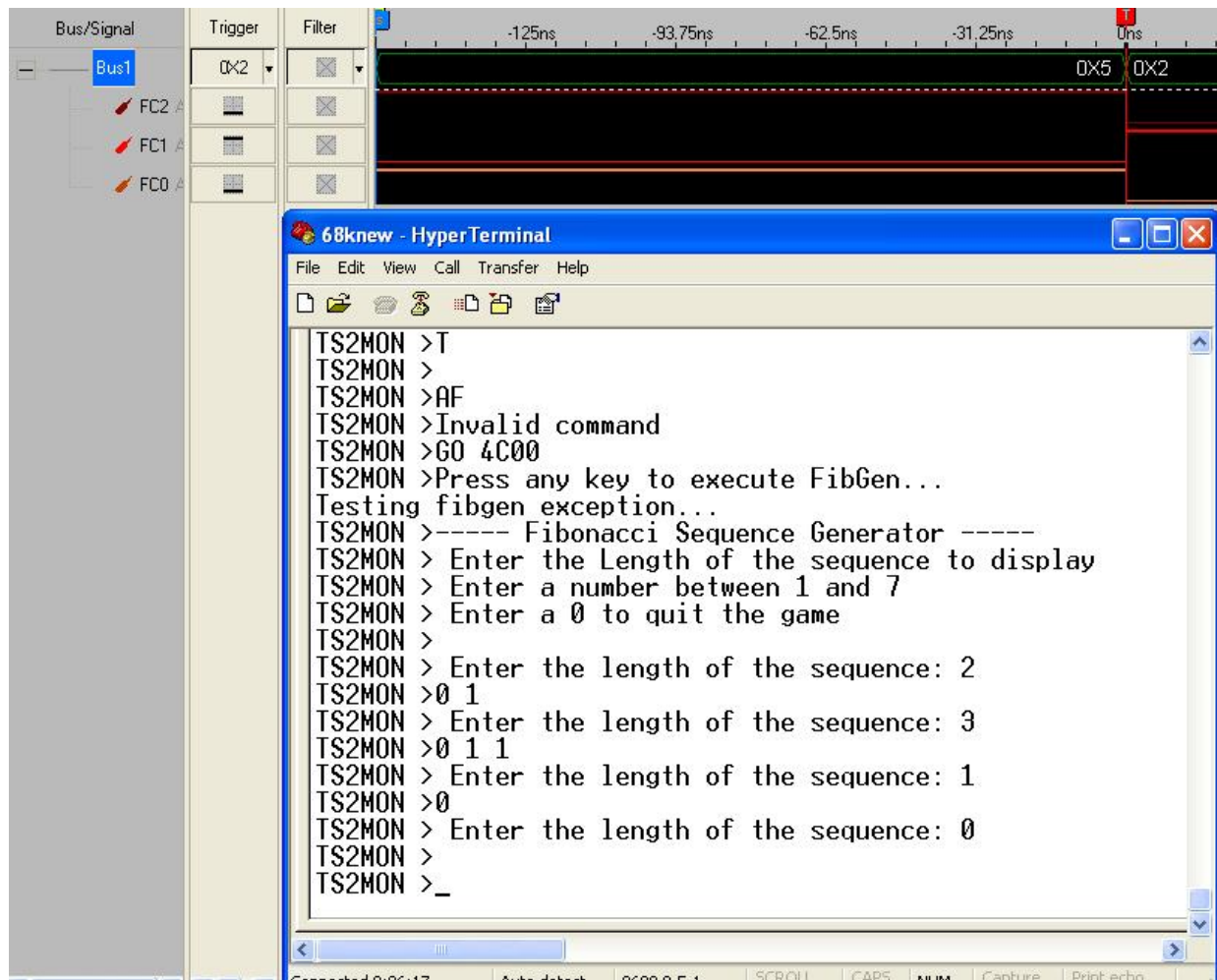


Figure 11. Operating States After Returning from Exception to User Program After Re-Engineering Firmware

Finally, after the user program has completed, control is returned to supervisor mode through a TRAP #15 call as the last line of the user program. This essentially works as a “return” statement. Proof of this functionality is shown in Figure 12 below.

```

68knew - HyperTerminal
File Edit View Call Transfer Help

TS2MON > Enter the length of the sequence: 1
TS2MON > 0
TS2MON > Enter the length of the sequence: 0
TS2MON >
TS2MON > DISPLAY
TS2MON >
TS2MON > Data reg      Address reg
TS2MON > 0 E05F0EA0    F0F00F0F
TS2MON > 1 8ED0B00F    0F0FF0F0
TS2MON > 2 780F0F0F    F0F0EAF0
TS2MON > 3 0F0FF0F0    47F0700F
TS2MON > 4 F0F00F0F    9C0F6AF0
TS2MON > 5 0F0FF0F0    1FF0E40F
TS2MON > 6 F0F00F0F    D90F13F8
TS2MON > 7 0F0FF0F0    3BF0D0AD
TS2MON >
TS2MON > SS = F40F0DF0
TS2MON > SR = 4EE0
TS2MON > PC = 784FC65F
TS2MON >
TS2MON >

```

Figure 12. Operating States while Returning from User Mode After Re-Engineering Firmware

The results of the operating mode proofs post re-engineering the firmware are shown below in Table 3.

Proof	Conclusion
Reset in Supervisor Mode	True
TS2MON Execution in Supervisor Mode	True
User application in Supervisor Mode	False
Interrupt in Supervisor Mode	True
Return from interrupt in Supervisor Mode	False
Return from user application in Supervisor Mode	True

Table 3. Operating Mode Proofs Post Re-Engineering Firmware

Part 3:

The M68k only has 16mb of memory that it can use. Part 3 involved expanding the amount of RAM by adding two AM9128-70CBD memory chips and associated control circuitry to an external breadboard, and connecting the circuit to the 68k's data, address, and bus mastering lines. This process involved placing the integrated circuit chips on the breadboard in an ideal layout and connecting them with minimal-length and color-coded wiring. This allowed for easy debugging and reduced electrical noise. An LED was also added to visually ensure desired operation. An over-head view of the memory expansion circuit is shown below in Figure 13.

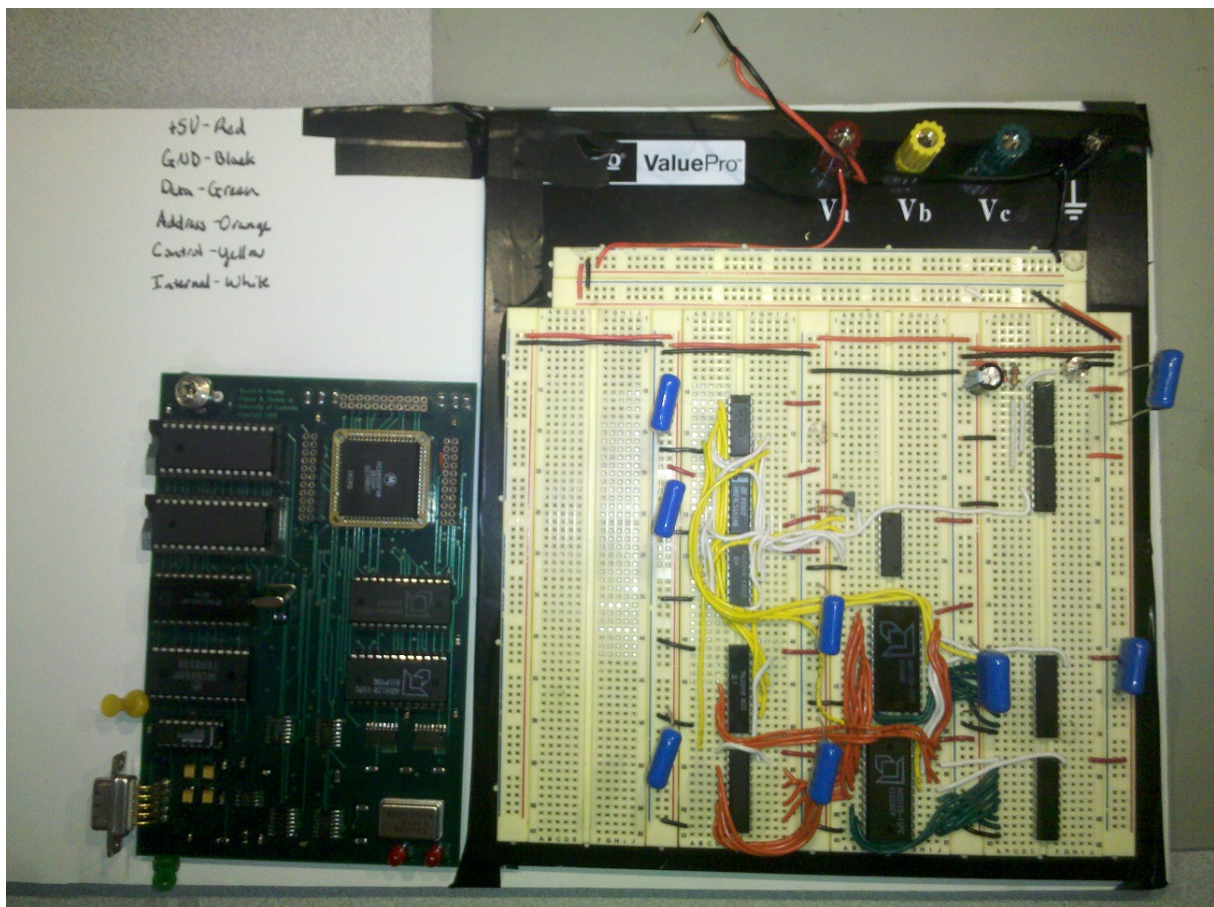


Figure 13. Wired Memory Expansion Unit and Reset Button

Parts Lists

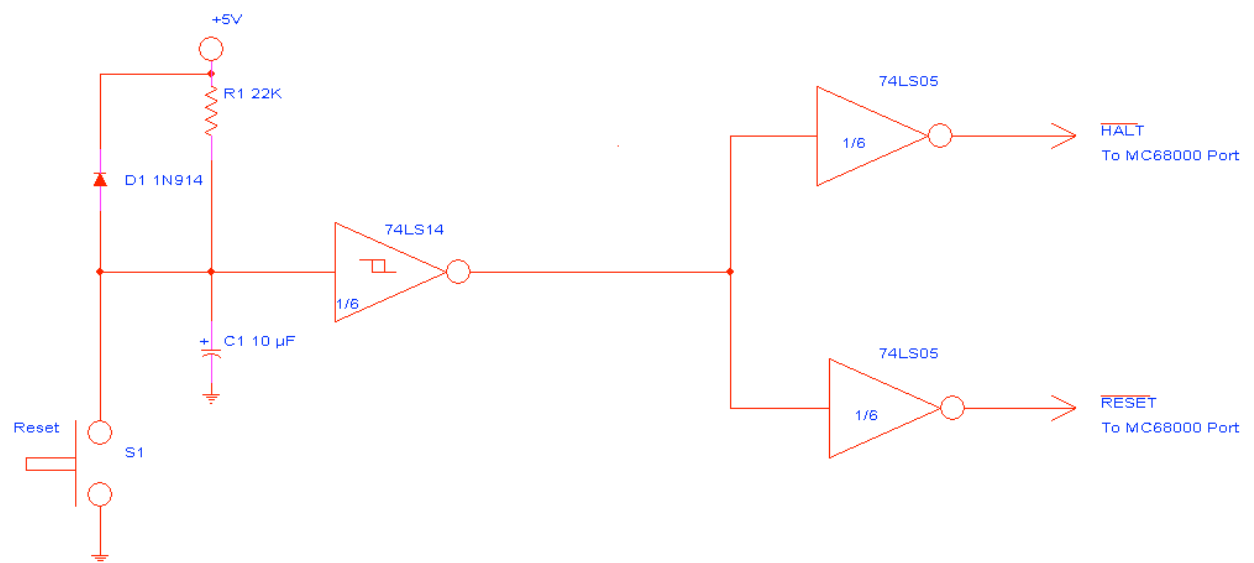
RESET circuit:

Qty.	Part#/Value	Description
1	22K Ohm	Resistor
1	1N914	Small Signal Switching Diode
1	10uF	Electrolytic Capacitor
1	SPST Switch	N.O. Push Button Type
1	74LS14	Schmitt Trigger Hex Inverter
1	74LS05	Open Collector Hex Inverter

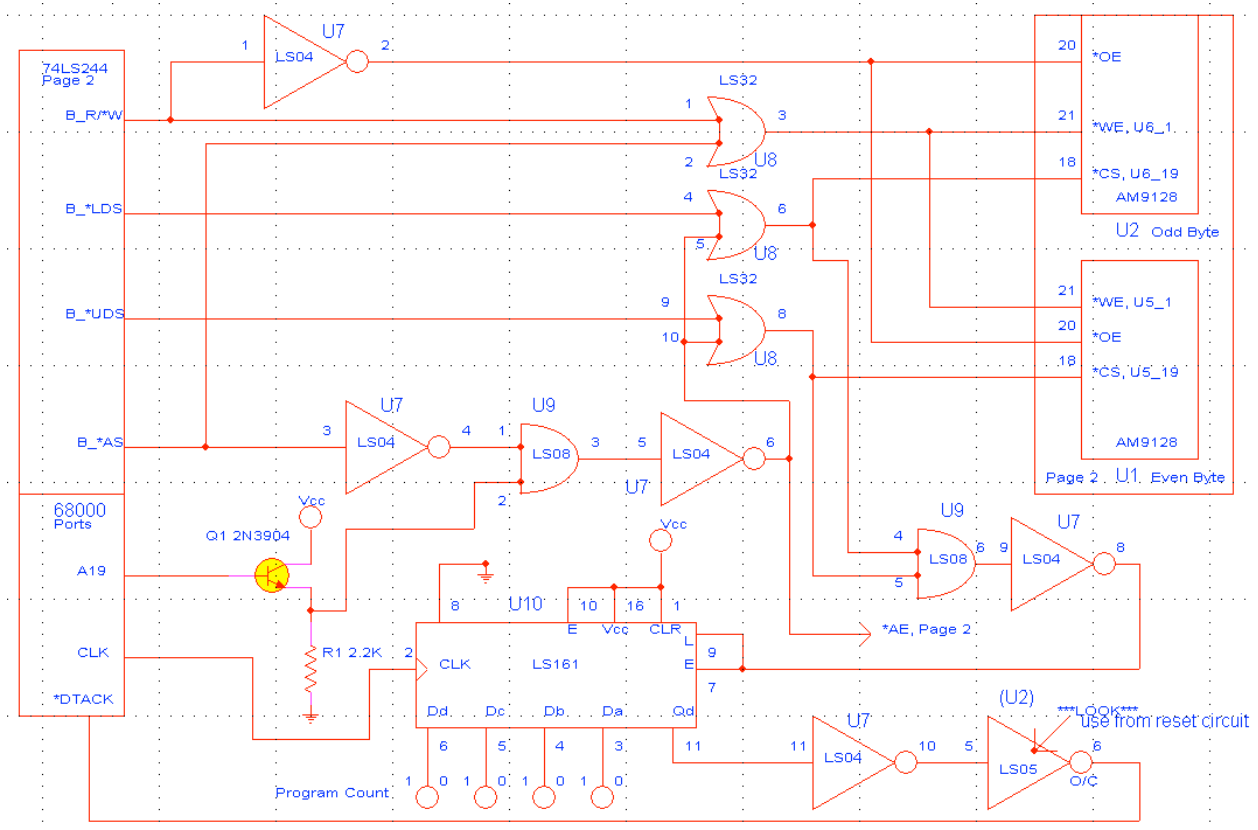
RAM Expansion circuit:

Qty.	Part#/Value	Description
2	74LS244	3-State Octal Buffer/Line Driver
2	74LS245	Octal Bus Transceiver
2	AM9128-150	2Kx8 Static RAM
1	74LS161	Dual D Flip-Flop
1	74LS08	Quad AND Gate
1	74LS04	Hex Inverter
1	74LS32	Quad OR Gate
9	0.1uF	Radial Capacitor
1	2N3904 or 2N2222	NPN Transistor
1	2.2K ohm	Resistor

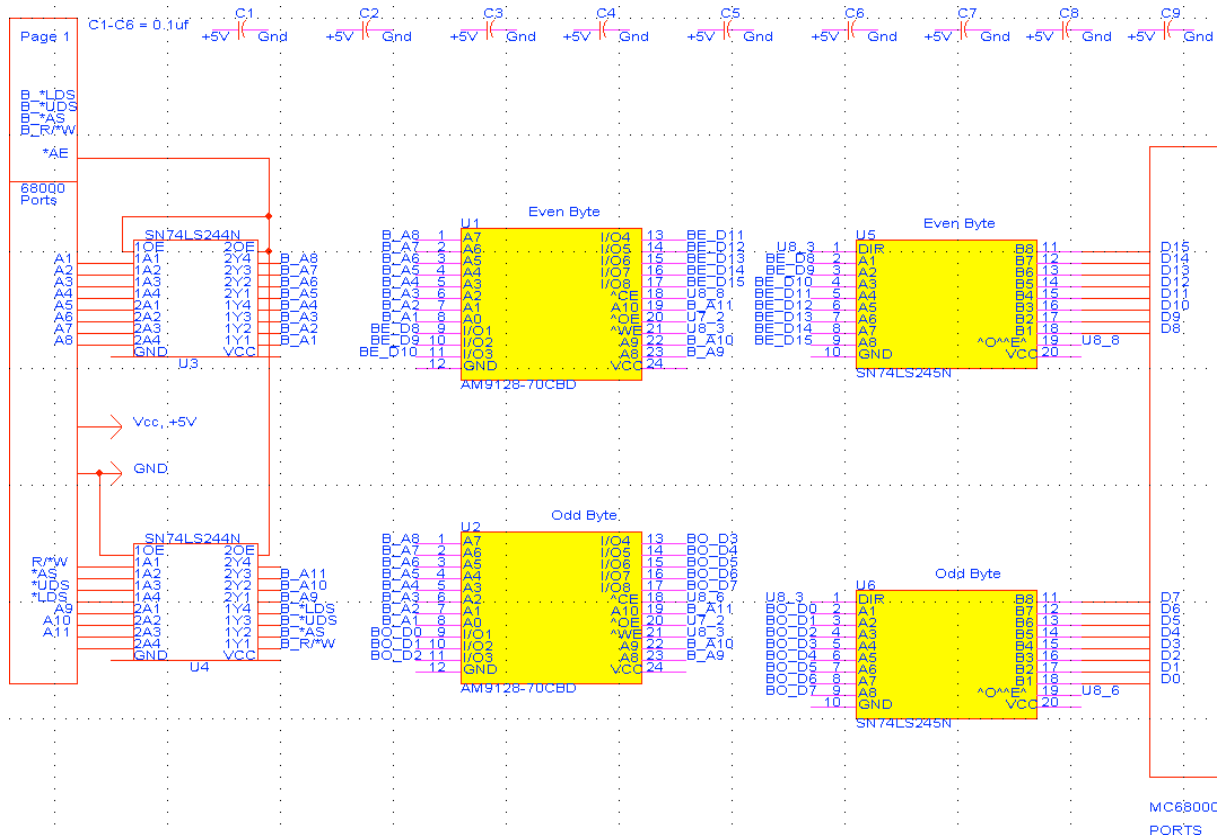
RESET Schematic



RAM Expansion Schematics



RAM Expansion Schematics (cont)



Part 4:

The serial port on the host PC is configured as Data Terminal Equipment (DTE) and the MC68000 minimal computer is configured as Data Communication Equipment (DCE). In this part of the lab it was determined, through the use of the M16C schematic, an ohmmeter and an oscilloscope, that the M16 QSK RS232 serial interface is also configured as DCE. Because the M16 QSK is DCE the same straight cable used for interfacing to the 68K minimal computer was used to connect the M16 QSK to the host PC via HyperTerminal.

Only one modification of the HyperTerminal configuration used with the 68K minimal computer was required in order to communicate with the M16: the parity option was changed from even to none (all other options including baud rate and number of data/stop bits are the same for both devices).

In addition, the M16 user program required two modifications. First the UART interrupt routine was modified in order to store 9 characters received over a serial line and print them to the lcd. Second the user program was modified so that a button press sends what characters are on the screen over the serial line.

Below in figures 17, 18 and 19 are depicted the M16 QSK schematic of the RS232 interface, a screen capture of HyperTerminal showing the host PC communicating to the M16 as well as a photograph of the M16 showing the message from HyperTerminal on its LCD.

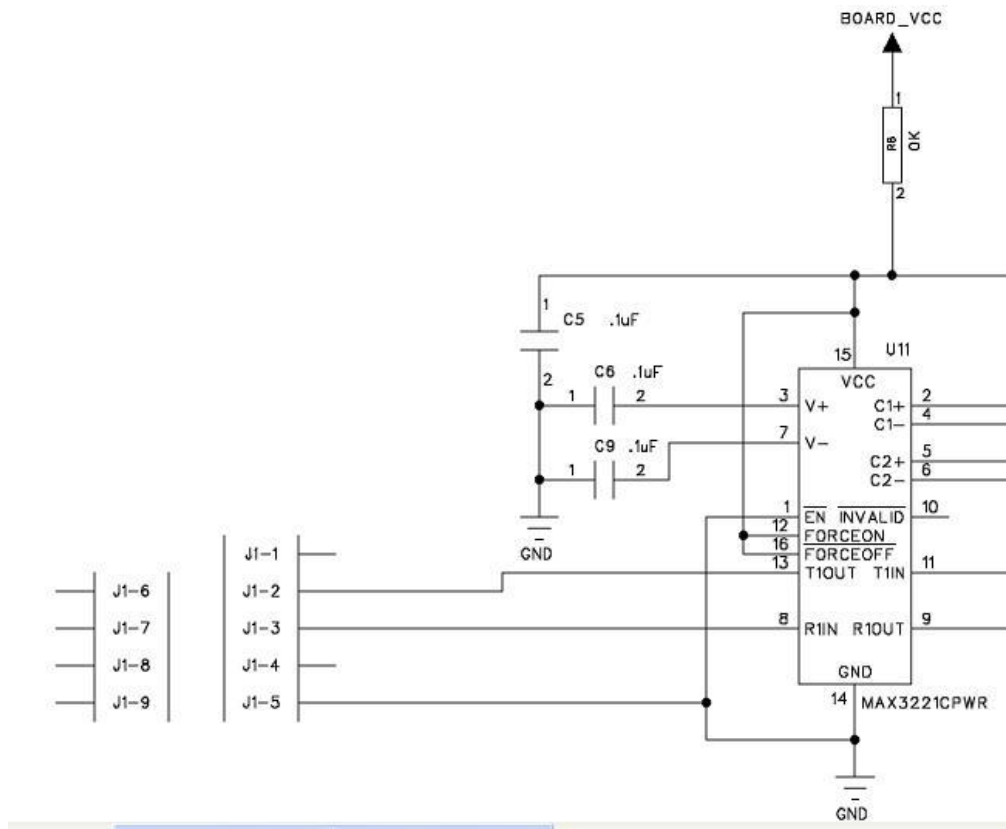


Figure 17. Schematic of M16 RS232 Circuit showing DCE configuration

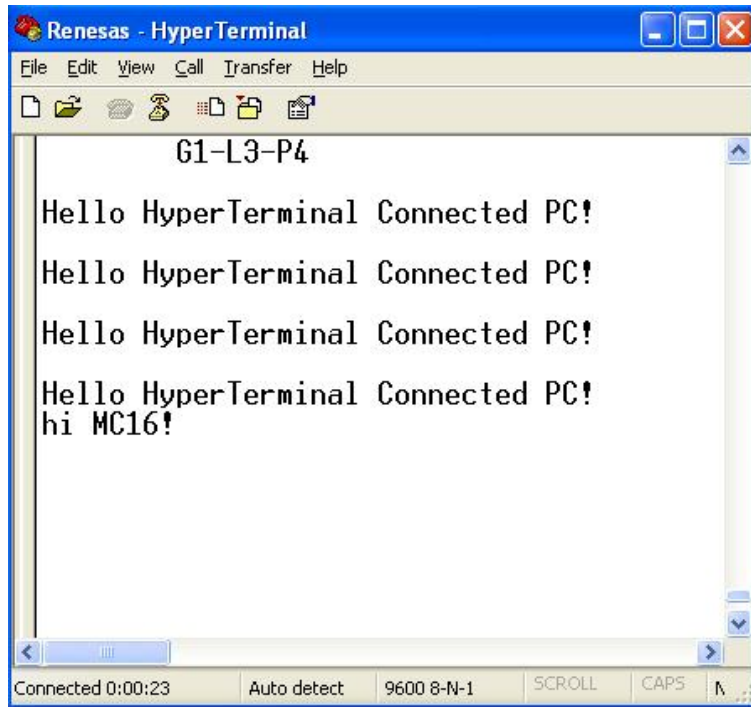


Figure 18. Screenshot of HyperTerminal Communicating to MC16 QSK via RS232 interface

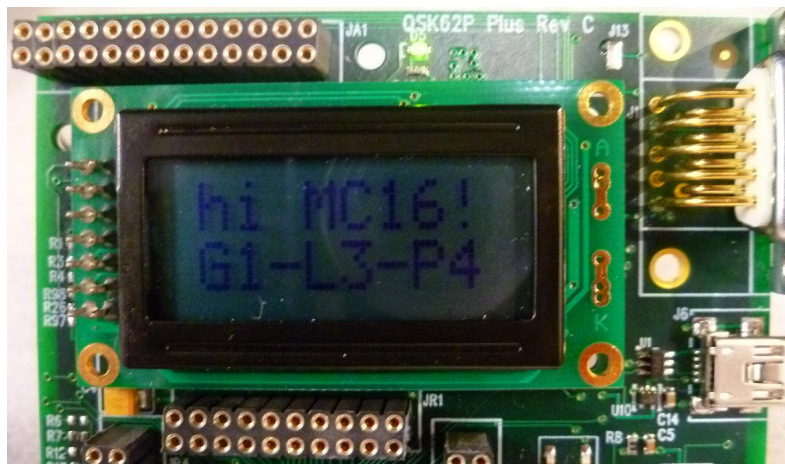


Figure 19. Photograph of M16 QSK LCD displaying message sent from host PC via RS232

Software

Part 2: Modified GO Routine

```
*
*****
*
* GO executes a program from a supplied address
*
GO      BSR      PARAM    Get entry address (if any)
        TST.B D7          Test for error in input
        BEQ.S G01        If D7 zero then OK
        LEA.L ERMES1(PC),A4    Else point to error message,
        BRA      PSTRING    print it and return
G01     TST.L D0          If no address entered then
        BEQ.S G02          jump to displaying an error message
        MOVE.L D0,-(A7)      push execution address to stack
        MOVE.W #$0700,-(A7)  push SR (set user mode, disable interrupts) to stack
        BRA.S RESTORE    execute program
G02     LEA.L GOERR(PC),A4    load GO error message (no address supplied)
        BSR  HEADING        branch to heading subroutine to print message
        RTS                RTS from BSR to G0, this give us command line after erro
RESTORE RTE      Return from exception to run program
```

Part 3: User program loaded into expanded RAM at \$80000 and used to verify switch to user mode and back again

```
START   ORG  $80000      * starting location of code in RAM (required)
        LEA.L PRESSK(PC),A4 * Ask user to press any key
        MOVE.L #4,D1
        TRAP #0
        MOVE.L #2,D1      * Newline
        TRAP #0

        CLR.L D0          * Wait for user to hit a key before continuing
        MOVE.L #0,D1
        TRAP #0

        LEA.L STR(PC),A4    * Tell user we're about to make an exception
        MOVE.L #4,D1
        TRAP #0
        MOVE.L #2,D1      * Newline
        TRAP #0

        MOVE.L #17,D1      * FIBGEN task
        TRAP #0

        TRAP #15          * Return statement (Exception - End of Program)

* Variables and Strings
PRESSK DC.B 'Press any key to execute FibGen...',0,0
STR     DC.B 'Testing fibgen exception...',0
END      START          * end of program (required)
```

Part 4: Modified Interrupt routine code and source for M16 to host PC RS232 communication

Modified Interrupt Routine:

```
int rx_count = 0;           // Initialize the input count to 0
char string[8] = "          "; // Initialize the char array to all spaces
void UART0ReceiveInterrupt(void)
{
    while(ri_u0c1 == 0); // make sure receive is complete

    while(ti_u0c1 == 0); // puts it in the UART 0 transmit buffer (echo input)

    u0tb = (char) u0rb;      // read in received data and echo out

    rx_count++;
    if(rx_count == 9 || (char)u0rb == '\r') { // If we have full input count
        BNSPrintf(LCD, "\t%c%c%c%c%c%c%c", string[0],
        string[1],
        string[2],
        string[3],
        string[4],
        string[5],
        string[6],
        string[7]); // print all chars in array
        rx_count = 0; // reset the input count to 0
        string[0] = string[1] = string[2] = string[3] =
        string[4] = string[5] = string[6] = string[7] = 0x20; // then reset characters to spaces
    }
    else {
        string[rx_count - 1] = (char)u0rb; // else load input character into array
    }
}
```

Main function from user program:

```
void main(void)
{
    MCUInit(); // Initialize the MCU
    InitDisplay("G1-L3-P4"); // Initialize the LCD and display "G1-L3-P4"
    InitUART(); // Initialize the UART (serial) interface
    BNSPrintf(SERIAL, "\n\r\tG1-L3-P4\n\r"); // Send initial message by serial intface
    TimerInit(); // Initialize Timers
    ENABLE_SWITCHES; // Enable the switches (bushputtons)

    BNSPrintf(LCD, "\tHyperTst"); // Print the name of the program to the LCD

    while(1) { // Loop forever!
        if(S1 == 0){ // If button is pushed send message to host PC by serial
            BNSPrintf(SERIAL, "\n\rHello HyperTerminal Connected PC!\n\r");
        }
    }
}
```

Analysis

In this lab, experience was gained in mapping pins on a development board, taking advantage of the 68k's user and supervisor modes, adding, interfacing, and using external RAM chips, and communicating with the Renesas micro-controller through a serial port. When it was first released, the 68k's two operating modes (user and supervisor) were a novel concept. Taking advantage of those modes gave an added layer of security and functionality to the modified firmware. No program in user mode can access supervisor-specific programs without calling an exception.

Interfacing with external RAM allowed for an expansion of available RAM and also gave valuable experience with using the 68k's data and address lines to control hardware peripherals. This knowledge is valuable in the field of embedded systems because embedded systems are all about interfacing microcomputers with the newest technologies.

An understanding of serial communications was developed by setting up the M16 to communicate with hyperterminal, the second system set up to do so. Doing so demonstrated, first, an understanding of DTE vs DCE and serial protocol in general and, second, and understanding of and ability to modify the M16 user program.

Conclusion

Here in this lab, skills were developed to ensure secure firmware operation, to expand and debug the hardware of the minimal computer, and to utilize serial communication with the M16. These skills will be valuable both for future labs in this course and for microcontroller programming in general.

The 68K platform includes built-in tools to separate secure modes of operation from less-secure modes of operation. Now and in the future, firmware, exceptions, and TRAP calls will maintain the use of supervisor mode while—with the modification of the GO routine—user programs will execute with only user mode privileges. This adds an important (but incomplete) layer of security against accidental or malicious harm from loaded programs.

So far, all work on the 68K has largely been independent of the hardware. Now, however, all of the pins available on the development board have been mapped and nearly all of them used in order to expand RAM, to add a hardware interrupt, and to debug the supervisor and user modes. Both the RAM expansion and the hardware reset required proficiency with breadboard wiring, including best practices to avoid EMI noise and to create easily readable circuits. Expanding RAM required proficiency with the process of address decoding. This means that in the future, debugging existing hardware and even adding novel hardware will operate smoothly.

Finally, by successfully demonstrating 2-way serial communication between the M16 and hyperterminal, a powerful tool has been developed. It is now possible, in practice, to exchange information between hyperterminal and both systems used so far in development. It is even possible, in theory, to do so between the M16 and 68K. Furthermore, in the event that new systems must be interfaced with the 68K or M16, the understanding of serial communications demonstrated here, will be leverage to great effect.

In all, the ability to expand and debug hardware, the use of secure firmware practices, and the ability to leverage serial communications—each of which was successfully demonstrated here—will prove crucial both in the future academic projects and in professional embedded systems design.

References

CECS 525 Project 3 Report Template by Eugene Rocky

The 68000 Microprocessor Fifth Edition by James L. Antonakos