1999

# A Scalable Multiprocessor

Charles Yarbrough

# A Scaleable Multiprocessor


## By: Charles Yarbrough

**Advisor: Dr. B. Earl Wells**


**Spring 1999**


**University of Alabama in Huntsville**


**Honors Senior Project**

## Abstract

To fulfill the requirements of a Senior Design project in Computer Engineering, I constructed at multiprocessor computer built on the Motorola 68000 architecture. This system is designed as a daisy-chain model with successive processors sharing a bank of memory for interprocessor communication. Each processor in the chain controls the reset on the following processor and is responsible for loading the shared memory and releasing its slave processor to run. This design is scaleable to any size by simply adding another shared memory bank and processor onto the end of the chain.

All I/O associated with the system is controlled by the master processor at the top of the chain. This board includes LED displays, DIP switches, and a serial port. Slave processors (CPU 2 and below) are intended to be much simpler, only accessing two or three banks of memory.

This design should show performance benefits with programs that are easily divisible and do not require much I/O from the lower processors. Any communication from slave processors must be passed between every intermediate CPU until it reaches the master.

At this point, the system contains two processors, both of which are Motorola 68000s running at 8MHz. Future plans include adding additional processors on to the daisy chain and writing a chess program to utilize all available processors.

# Table of Contents


Introduction 1

Design Overview 1

CPU 1 1

CPU 2 4

Design Implementation 4

CPU 1 5

Reset Circuitry and the Processor, Itself 5

Bus Buffering 6

Address Decode Logic 6

DTACK' and BERR' 8

EPROM and Dedicated SRAM 9

Miscellaneous Input and Output 9

Shared Memory 10

CPU 2 Control 11

CPU 2 12

Address Decoding Logic 12

Expandability 12

Progress to Date 13

Future Work 13

Bibliography 14

Appendix A-1

## Introduction

This paper describes my design and implementation for a scaleable multiprocessor computer based on the Motorola 68000 architecture. The initial system will contain two 68000 central processing units (CPUs) on a single backplane. The two processors will have dedicated memories and/or devices, and they will share a single bank of memory to use for inter-processor communication. The first CPU will be responsible for all input and output (I/O) from the board, so in addition to dedicated banks of EPROM and SRAM, it will have a pair of 8-segment LEDs, a bank of DIP switches, and a serial port. It will also have direct control of the second CPU's reset circuitry. The second CPU, at this point, will only have a dedicated SRAM, although it can be expanded to have access to a third CPU and second shared memory.

This computer is my Senior Design Project for Computer Engineering. The system is not yet complete, so this paper will only cover the hardware and software complete at this time. The project, as a whole, will continue on beyond this semester and be expanded for the Honors Senior Project in 1999.

## Design Overview

### I. CPU 1

The first CPU is responsible for all I/O off of the board. It also has access to the only ROM on the system. Therefore, it must copy or load the appropriate programs to the volatile memories of each slave processor. The system communicates with users through several devices: the DIP switches for user input, the 8-segment LEDs for status information, and the serial port for high speed data transfers.

The 68000 architecture has a 16-bit data bus and 24-bit address bus. This allows for access to byte pairs, called words in an address space of 16 megabytes. All devices used are 8-bit devices, meaning that each only has 8 data lines. For proper memory access, the static RAMs (SRAMs) and EPROMs have to be used in pairs to achieve 16 accessible bits. The simple I/O devices only use 8-bits, and software access is limited to byte size transfers.

A. Memory Devices

EPROM is nonvolatile memory that can be erased with ultraviolet light and reprogrammed. This process is time consuming, taking approximately half an hour to erase each EPROM before it can be programmed again.

There is a single bank of EPROM accessible solely by the first processor. Two 128k x 8 bit EPROMs are tied together to split the data bus's 16 data lines. This ROM holds the initial Program Counter and Supervisor Stack Pointer that the 68000 reads on bootup. It also contains the main program for the system. When the bootloader is complete, the EPROM will hold code telling the first processor to read the serial port and copy a new program into the SRAM of both processors. This will allow new software to be executed without reburning the EPROMs.

SRAM is volatile memory meaning that it will lose its contents when power is disconnected. The single bank of SRAM, like the EPROM, consists of a pair of 8 bit chips. Each chip is 32k bytes. The bootloader will be stored in the EPROM and copy new programs from the serial port to the SRAM for execution.

The SRAM is also used for the stacks. These are stored variables that are necessary for such functionality as subroutine calls.

The shared memory will consist of a bank of two 8k x 8 bit SRAMs. This will be the only place for the two processors to communicate with each other. Both CPUs will have access to this memory, but not at the same time. On startup, the first processor will copy the program for the second processor into the shared memory region, then release CPU2's halt circuitry. This will start CPU2 running the new code.

B. Input and Output Devices

Two 8-segment Light Emitting Diode (LED)s are used for displaying the board status. Each is addressable separately and can display most standard alphanumeric symbols.

A single bank of 8 Dual Inline Pin (DIP) switches is used for basic user input to the board. The processor can read an 8-bit byte from the switches and test the value to determine the user's input. These switches will be used to select from multiple programs stored in the EPROM at bootup.

A 16550 UART is used for off-board serial communication. This serial device is only 8-bits wide, so transfers are limited to byte sized. The serial port is the central device in the bootloading process. The multiprocessor system will be connected to a personal computer through a serial line. New software will be downloaded through the serial port and copied into SRAM where it will be executed.

C. CPU2 Control

The first processor will have control of the second processor's halt line. Until it has finished loading the second processor's memory, the first CPU will hold the second's halt line. When this halt line is released, CPU2 will begin executing the code loaded in the shared memory bank.

II. CPU 2

A. Memory Devices

CPU2 will treat the shared memory bank as its boot strap memory and will expect its program code to reside here. This may cause some performance lags if CPU1 polls this shared memory while CPU2 is running.

In order to reduce contention for the shared memory, CPU2 will have a dedicated SRAM that it can either copy its data and program into or use simply as stack space. This bank of memory will consist of two 32k x 8 bit SRAMs.

B. Optional – CPU3 control and Shared Memory 2

This design is expandable to any number of CPUs by adding them on to the end of the daisy chain. Each processor will have a shared memory with the processor on each side in the chain. A given CPU will also have Halt control of the next-lower CPU.

**Implementation**

The following sections give a detailed description of the on-board implementation. See the appendix for diagrams showing circuit layout.

## I. CPU 1

The first processor consists of a Motorola 68000 microprocessor rated at 10 MHz, reset circuitry, miscellaneous logic, several I/O devices, and banks of EPROM and SRAM.

### A. Reset Circuitry and the Processor Itself

In order to properly clear the 68000 on startup and to reset it during operation, a "monostable vibrator" must be used to pull the HALT and RESET lines low for at least 100ms (1: 357). I used a 555 timer with a resistor of 1MΩ and a capacitor of 0.47μF. This gave a time constant of 0.47 seconds, or 470ms – sufficient time to reset the CPU.

The 68000 can also output HALT or RESET signals if necessary, to drive the board into a halted state. Any logic used as inputs to the HALT or RESET pins must be buffered, so that two outputs are not tied together. Open Drain CMOS buffers (74HC07) were used to safely drive these two bi-directional lines on the CPU.

A Light Emitting Diode (LED) is connected to the HALT line to the CPU to detect when the system is HALTed. This LED has helped tremendously in debugging the system because it indicates when the system stopped due to program errors.

An 8 MHz oscillator drives the entire system. These 68000s are rated for 10 MHz, but using a slower speed reduces power consumption and leaves room for error in device timing. An 8MHz system speed allows for somewhat slower devices to be used, but the slowest memory on the system is rated for 12 - 16

MHz. This also adds to flexibility in device timing. In order to isolate the CPU's clock from the rest of the board, the clock signal is split on two inverters. Isolating the CPU's clock yields a cleaner signal by reducing noise from other devices. Refer to Appendix A-1 for RESET and Clock circuitry.

Several features of the 68000 are not used on this system. These include interrupts, function codes, and 6800 peripheral bus arbitration (5: 3-1). All input lines associated with these functions are tied high through 4.7 kΩ resistors.

B. Bus Buffering

This system requires a large number of chips, so the address, data, and control lines drive many logic gates. To protect the 68000 from line spikes and high current, an array of parallel buffers is set up on the entire address and data bus. Additionally, several control lines are buffered, including the data strobes, LDS' and UDS', the address strobe, AS', and the Read/Write signal. The data lines are bi-directional, so 74HC245 logic chips are used to buffer D0-D15. An unbuffered Read/Write signal is used to control dataflow direction across these buffers. The address and above mentioned control lines are unidirectional, so 74HC244s are used to buffer A1-A23 as well as LDS', UDS', AS', and R/W'.

C. Address Decoding Logic

The Motorola 68000 uses memory mapped I/O to access all of its peripherals at specific locations in a single address space. To differentiate between devices, a 3-8 decoder (74HC138) is used with A21, A22, and A23 as its input lines. This divides the 68000's 16 megabytes of addressable space (7: 4) into 8 equal divisions of 2 megabytes each. The output lines of the decoder are used as

enable triggers on each device. This partial address decoding wastes address space, since every device was smaller than 2 megabytes in size (see Table 1). The method does, however, make address decoding simple and minimizes the chip count.

**Table 1: Memory Map of CPU 1**

| Device | Address Range (Hexadecimal) |
|---|---|
| SRAM 1 (Dedicated) | 000008 - 1FFFFF |
| SRAM 2 (Shared) | 200000 - 3FFFFF |
| LED 1 | 400000 - 5FFFFF |
| LED 2 | 600000 - 7FFFFF |
| DIP Switches | 800000 - 9FFFFF |
| Serial Port | A00000 - BFFFFF |
| EPROM | 000000 - 000007, C00000 - DFFFFF |
| CPU 2 Control | E00000 - FFFFFF |

On startup, the 68000 expects to find a Supervisor Stack Pointer (SSP) and Program Counter (PC) in the first two long-words of memory (addresses $000000 - $000007). It also expects the exception vectors to be located immediately above in addresses $000008 - $0003FF. The SSP and PC must be stored in non-volatile memory, such as the EPROM, but the exception vectors need to be changed with new programs (2: 1). To map the EPROM into the first 8 bytes of memory, a complex address decoding logic is used to mirror this range from the EPROM's original address, $C00000 - $C00007. SRAM resides at the address range $000000 - $1FFFFF, but the first 8 bytes of this range are forced to reference the EPROM. See the circuit diagram on A-2.

## A. DTACK and BERR

The 68000 is an asynchronous processor, which means that it waits for confirmation after each read or write cycle. The CPU knows a data transfer is complete when it receives a Data Transfer Acknowledge (DTACK'). After the Address Strobe is applied, the processor expects DTACK' to occur within 2 state periods (1 clock cycle). With an 8MHz clock, one clock cycle is 125ns. If a DTACK' does not occur in the expected time, wait states are inserted one clock cycle at a time until DTACK' or a Bus Error (BERR') is detected. BERR' is a "watchdog timer" that keeps the processor from waiting forever for DTACK'. In this system, BERR' is generated with 4 flip flops arranged in series, clocked with E and activated by the presence of AS'. E is an external clock used for 6800 peripherals. It's cycle time is 10 times longer than that of the system clock – in this case, E is 800 kHz. When AS' goes active, the flip-flops pass an input value to each other sequentially at 800 kHz. If DTACK' has not occurred by the time the last flip-flop outputs its signal, a Bus Error is flagged to the CPU. With 4 flip-flops running at this clock rate, BERR' will go active in 5 ms.

All devices and memory dedicated to CPU1 are fast enough to allow DTACK' to be signaled as soon as the device was activated. The only slow device is the shared memory. Though the memory is not slow in itself, access can not be guaranteed immediately. In this case, DTACK' is generated by the granting signal to the shared SRAM bank.

B. EPROM and Dedicated SRAM

The EPROM (Erasable Programmable Read Only Memory) is a bank of non-volatile memory that can hold a permanent copy of the computer's program. In the final system, the EPROM will contain a bootloading program that will read the Serial Port and download new programs for execution. As mentioned earlier, the first 8 bytes of the EPROM is mirrored down to the bottom of memory because it holds the SSP and PC.

The SRAM is loaded in the bottom block of memory, but the usable space starts at $000008. This volatile memory can not permanently store a program, but could be used to run a program once loaded. This bank of SRAM can contain the exception vectors, and stack memory, as well as any bootloaded programs or data.

The system contains 256 Kbytes of EPROM in one bank. The 68000 has a 16-bit data bus, so two 128 Kb x 8 bit EPROMs are used in parallel. Similarly, two 32 Kb x 8 bit SRAMs are used to create a dedicated bank of 64 Kbytes.

C. Miscellaneous Input / Output

Two 7-segment (plus decimal point) LED displays are used as the primary output device on the system. These displays can be used to show system status, or to output specific memory holdings. They are controlled with octal flip-flop arrays (74HC574). The output controls of these arrays are tied low so that the displays remain on continuously. To decrease the current flowing through these LEDs, a 330 Ω resistor is placed on each output pin between the flip-flops and the LED segment. For a 1.5 V LED, this drops the current down on a 5 V CMOS output to approximately 10 mA.

The primary board-level input device on the system is an array of 8 Dual Inline Pin (DIP) Switches. The values on these switches can be read by the processor and interpreted to determine a course of action. A 74HC245 buffer is used to tie closed switches high and leave open switches low.

The serial port consists of a 16550 Universal Asynchronous Receiver / Transmitter (UART) and a 145407 Line Driver. The 16550 is an 8-bit wide serial controller with internal registers. These registers have to be properly set to determine transfer speed and terminal settings before sending or receiving data (6:1). The line driver amplifies the TTL outputs of the 16550 for transmission over a serial cable.

The serial port is necessary for bootloading new programs on to the system. These new programs are sent in the form of S-Records, that are ASCII readable and consist of the hex codes representing program (4: C-1). A program stored in the EPROM reads the serial port and downloads an S-Record, translates it, stores it in the SRAM, and executes it.

D. Shared Memory

Interprocessor communication is done through a single bank of shared SRAM. Both CPU 1 and CPU 2 have access to this memory with an arbitration circuit and buffering to separate the two sets of buses.

1. Two sided Buffering

To keep from corrupting the address, data, or control bus of either processor, two sets of buffers are used with mutually exclusive enabling to allow access to the shared bank. For convenience, 74HC245s are used to

buffer all of the lines instead of trying to use unidirectional 74HC244s for address and control. To minimize the chip count, the size of the shared memory is reduced to a pair of 8 Kbytes x 8 bit chips. These only used 13 address lines (3:1), leaving 3 available lines for LDS', UDS', and R/W'.

2. Arbitration Logic

A rather simple arbitration circuit prevents both processors from accessing the shared memory at the same time. The only possible problem with this implementation is that it does not account for both processors requesting the shared memory at EXACTLY the same time. To reduce this risk, a series of buffers (74HC07)s offset the second CPU's clock by at least 20 ns from CPU 1 (8: 3). This delay should prevent memory accesses from occurring before an output could be generated on the arbitration circuitry.

E. CPU 2 Control
*** THIS DOES NOT YET WORK AND WILL BE CHANGED. ***

The final device on CPU 1's memory map is a flip flop (74HC74) that controls the HALT' line to CPU 2. On startup, CPU 1 would lower this line a HALT CPU 2. It can then load the shared memory with an appropriate program and release CPU 2 to run when ready. The speed of device accesses and the fast bus cycle of the system means that to properly reset CPU 2, the first processor has to hold the reset flip-flop low for at least 100 bus cycles (9: 1). To do this, a 555 reset timing circuit is included with CPU 2, as in CPU 1. Since it may be possible

for CPU 2 to keep running even if CPU 1 halts, the HALT' lines from the two processors are NOT tied together.

## II. CPU2

The second processor has a much simpler implementation than the first in that it has no I/O or EPROM. Its shared memory contains all necessary programming to get it running. A second bank of dedicated SRAM is included to either copy the program to or use as stack space to reduce contention for the shared memory.

### A. Address Decoding Logic

It is not necessary to mask sections of memory around in CPU 2's memory map, as it was with CPU 1's EPROM. CPU 2 expects both the bootstrap information (SPP and PC) and the exception vectors to be in the shared memory. A 74HC139 2-4 decoder divides up device space, so each device gets 4 Mbytes, instead of 2 Mbytes like in CPU 1 (10: 1). See Table 2.

**Table 2: Memory Map of CPU 2**

| Device | Address Range (Hexadecimal) |
|---|---|
| SRAM 1 (Shared) | 000000 - 3FFFFF |
| SRAM 2 (Dedicated) | 400000 - 7FFFFF |
| SRAM 3 (Shared) – Optional | 800000 - BFFFFF |
| CPU 3 Control – Optional | C00000 - FFFFFF |

### B. Expandability

CPU 2 uses a 2-4 decoder (74HC139) for device address decoding, and uses the first two lines for the implemented memory. The third and fourth lines can be used for a third processor and second bank of shared memory (10: 1). Though it would not be necessary to use the 68000 as an additional processor on

the system, issues such as Endian standards and different programming models would make it difficult to implement a non-680x0 family processor.

**Progress To Date**

As of this paper (April 13, 1999), the first processor is completely finished. The second CPU is complete except for the dedicated SRAM, which is not necessary for shared memory arbitration tests. The reset circuitry for the second CPU has been rewired from the original project to include a 555 timer circuit, but it is still unreliable.

The clock on the second CPU will be inverted from the first CPU to guarantee that two requests will not reach the shared memory arbitration logic at the same time.

A software test case was generated where the first processor copied the program counter, supervisor stack pointer, and a counting program to the shared memory for CPU2 to execute. CPU2 was then reset and runs the program in shared memory. The program has the processor count to $100003 and then increment an 8-bit counter in the shared memory. Meanwhile, CPU is counting to $50000 and incrementing its own counter. It displays its counter value on LED1 and the value of CPU2's counter in shared memory on LED2. See the attached assembly code for this program.

**Future Work**

This project will be continued and expanded beyond this semester. Any incomplete software test cases will be written, as well as parallel processing benchmark and application programs. I hope to add a third processor to verify the scalability ease. My final goal is to produce a parallel processing chess program utilizing all available processors.

## Bibliography


1. Antonakos, James L. The 68000 Microprocessor. 3rd ed. Englewood Cliffs, New Jersey: Prentice Hall; 1996.

2. Brown, Geoffrey; Harper, Kyle. MC68008 Minimum configuration system. Motorola Application Note AN897. 1984.

3. HM6264BI Series, 8192-word x 8-bit high speed CMOS static RAM. Hitachi Datasheet; 1996 September.

4. M68000 Family programmer's reference manual. Phoenix, Arizona: Motorola Literature Distribution; 1992.

5. M68000, Microprocessor user's manual. 9th ed. Phoenix, Arizona: Motorola Literature Distribution; 1993.

6. PC16550D universal asynchronous receiver / transmitter with FIFOs. National Semiconductor Datasheet; 1995 June.

7. Starnes, Thomas W. Design philosophy behind Motorola's MC68000. Byte. 1983 April. Reprinted by Motorola as AR208.

8. TC74HC07AP/AF, Hex buffer. Toshiba Datasheet; 1997 August.

9. TC74HC74AP/AF/AFN, Dual D-type flip flop preset and clear. Toshiba Datasheet; 1997 August.

10. TC74HC139AP/AF/AFN, Dual 2-to-4 line decoder. Toshiba Datasheet; 1997 August.

```
*; Shared RAM test program 2
*; CPE 437: Senior Design
*; By: Charles Yarbrough
*; Spring 1998
*; Motorola 68000 Multiprocessor

        ORG     $00000000
        DC.L    $00001000       ; stack pointer
        DC.L    $00C00008       ; program counter
        ORG     $00000008

        move.l  #$00C00500,($00000008)  ; bus error exception handler
        move.l  #$00400001,A0           ; LED 1
        move.l  #$00600001,A1           ; LED 2
        move.l  #$00800001,A2           ; DIP switches

        move.b  #$25,(A0)         ; three horizonal lines
        move.b  #$25,(A1)         ; three horizonal lines

        move.b  #$00,($00E00000)    ; halt CPU 2

        move.l  #$00001000,($00200000)  ; CPU 2 Stack Pointer

        move.b  #$48,(A0)         ; "1" on LED 1

        move.l  #$00000400,($00200004)  ; CPU 2 Program Counter

        move.b  #$3D,(A1)         ; "2" on LED 2

        bsr     copyp           ; copy program to CPU 2

        move.b  #$6D,(A0)         ; "3" on LED 1
        move.b  #$4B,(A1)         ; "4" on LED 2

        bsr     wait1

        move.b  #$FF,($00E00000)    ; restart CPU 2

        move.l  #$00000000,D1
add0    move.l  #$00000000,D0
add1    add.l   #$00000001,D0
        cmp.l   #$00080000,D0
        bne     add1

        add.b   #$01,D1
        move.b  D1,(A0)
        move.b  ($00201500),(A1)
        cmp.b   #$F0,D1
        bne     add0

        move.b  #$7E,(A0)         ; "O" on LED 1
        move.b  #$5B,(A1)         ; "K" on LED 2

done    bra     done

wait1   move.b  (A2),D0
        cmp.b   #$01,D0
        bne     wait1
        rts

copyp   move.l  #$00200400,A4
```

```
        move.l  #$00C00400,A3
        move.b  #$00,D2
incr    add.b   #$01,D2
        move.l  (A3),(A4)
        add.l   #$00000004,A4
        add.l   #$00000004,A3
        cmp.b   #$0A,D2
        bne     incr
        rts


        org     $00000400       ; CPU2's program
        move.l  #$00000000,D1
add2    move.l  #$00000000,D0
add3    add.l   #$00000001,D0
        cmp.l   #$00100003,D0
        bne     add3

        add.b   #$01,D1
        move.b  D1,($00001500)
        bra     add2

        org     $00000500   ; bus error exception handler
        move.b  #$73,(A0)   ; "b" on LED 1
        move.b  #$37,(A1)   ; "E" on LED 2
        rte                 ; try again
```