

Contents

1	Introduction	1
2	Overview of the MPP Project	3
2.1	Why Use the 68HC11?	3
2.2	Applications of the MPP	4
2.3	Software Development on the MPP	5
2.4	Learning to Program the 68HC11	7
2.5	Description of the MPP Hardware	7
2.6	MPP Default Memory Map	14
2.6.1	The ROM Error Message	16
2.7	The MPP Power Supply	16
2.8	Software Distribution	16
2.9	Setting Up Communications	17
2.9.1	The Dumb Terminal	18
2.9.2	Communication: Step by Step	18
2.10	A First Program on the MPP	19
2.10.1	Step 1: Write the program	20
2.10.2	Step 2: Assemble the program	20
2.10.3	Step 3: Upload the S19 file into the MPP board	21
2.10.4	Step 4: Check that the program loaded correctly	21
2.10.5	Step 5: Test the program	21
3	Interfacing Hardware	22
3.1	Output Control: First Steps	22
3.2	Input Data: First Steps	23
3.3	Input-Output Ports	23
3.4	Example: Transistor LED Driver	24
3.5	External Logic Driving an Input Port	26
3.6	Port Expansion using the Serial Peripheral Interface	26
3.6.1	Serial Peripheral Interface Routine	27
3.7	Memory Mapped Peripheral: D-A Converter	29
3.8	Memory Block Mapped Peripheral: Video Display Interface	30
3.9	Debugging a Memory Mapped Interface	33
3.10	The MPP Power Supplies	35

3.10.1	Using the 5 Volt Logic Supply	35
3.10.2	The MPP Raw DC Supply With A New Regulator	35
3.10.3	Stealing a Negative Voltage	36
3.11	Appendix: Estimating the Power Capability of the Regulator	37
4	Programming the Liquid Crystal Display	40
4.1	LCD Code	40
5	Buffalo Monitor: Technical Details	46
5.1	Summary of Resources used by the Buffalo Monitor	46
5.2	Interrupt Vectors under Buffalo	48
5.3	Buffalo Monitor Commands	50
5.3.1	ASM: Assembler-Disassembler	50
5.3.2	BF: Block Fill	50
5.3.3	BR: Breakpoint Set and Clear	50
5.3.4	BULK: Clear All EEPROM Locations	51
5.3.5	CALL: Run User Subroutine	51
5.3.6	GO: Start User Program	51
5.3.7	HELP: Display the HELP Screen	52
5.3.8	LOAD T: Upload S Records from Terminal Port	52
5.3.9	MD: Memory Display	53
5.3.10	MM: Memory Modify	53
5.3.11	MOVE: Move Block of Memory	53
5.3.12	P: Proceed	54
5.3.13	RM: Register Modify	54
5.3.14	T: Trace (Single Step)	54
5.4	The Monitor <i>Run</i> Command: Technical Details	55
5.4.1	Starting a User Program	55
5.4.2	Breaking From a User Program	56
5.4.3	RUNONE: The Single Step Facility	56
5.5	Buffalo Monitor Utility Routines	57
6	Microprocessor Arithmetic	58
6.1	Virtual Machine Instructions	60
6.2	Floating Point Mathematics	60
6.3	Floating Point Notation VS Fixed Point Notation	61
6.4	Lookup Table Method	62
6.5	Floating Point Calculations	62
6.6	The 68HC11 Floating Point Package	63
6.7	Locating the Floating Point Package	63
6.8	Floating Point Number Format	64
6.9	IEEE Floating Point Format	65
6.10	Interpreting Floating Point Numbers	65
6.11	The Floating Point Stack for Calculations	67
6.12	Floating Point Error Messages	68
6.13	Example Program: Hypotenuse Calculation	69

6.14	MATH11 Reference Material	72
6.14.1	Conversion Routines	72
6.14.2	Fundamental Mathematics Routines	72
6.14.3	Trigonometric and Polynomial Functions	73
6.14.4	Miscellaneous Functions	73
6.14.5	MATH11 Jump Table	75
7	Re-configuring the MPP Board	76
7.1	Setting the Default BAUD Rate	76
7.2	The 68HC11 Memory Map	77
7.3	Programming the Memory Map Logic	79
7.4	Modifying the Buffalo Monitor EPROM	80
7.4.1	An Example	81
7.5	Read and Write Visibility and Bus Conflicts	82
7.6	EPROM Size and Location	82
7.7	Memory Map Decoder	83
7.8	MPP Decoding	84
7.9	MPP Bus Timing	86
7.10	Read-Write Timing: Input and Output Registers	88
7.11	Read-Write Timing: Memory Devices	89
8	Single Chip Mode	92
8.1	Resources in Single Chip Mode	93
8.2	A Development Strategy	95
8.3	Software for Single Chip Operation	95
8.4	Hardware for Single Chip Operation	99
	Bibliography and References	102

List of Figures

2.1	Target and Development Systems	6
2.2	MPP Board	9
2.3	MPP Block Diagram	10
2.4	Jumper Positions and Functions	11
2.5	MPP Schematic, Processor Section	12
2.6	MPP Schematic, Memory Section	13
2.7	MPP Memory Map	14
3.1	68HC11 Electrical Characteristics Data Sheet	24
3.2	Port Equivalent Circuits	25
3.3	SPI Interface Example	27
3.4	Address and Data Bus Header JP15	30
3.5	Digital to Analog Converter Interface	31
3.6	Memory Mapped Periferal	32
3.7	Memory Bus Signals during Debugging	34
3.8	Adaptor Output Voltage	35
3.9	7 Volt Regulated Supply	36
3.10	MPP Heat Sink Dimensions	38
4.1	LCD Control Codes	41
5.1	Buffalo Monitor Interrupt Vector Jump Table	49
5.2	Monitor Utility Routines	57
6.1	MATH11 Floating Point Number Format	65
6.2	IEEE Floating Point Number Format	66
7.1	Memory Map for the 68HC11	78
7.2	1:8 Memory Map Decoder	83
7.3	PAL Memory Select Decoder	84
7.4	Memory Map Example, Truth Table	85
7.5	68HC11 Bus Waveforms	87
7.6	Memory System	90
8.1	68HC11 Single Chip System, Board Layout (Actual Size)	93
8.2	68HC11 Single Chip System, Schematic	94

Chapter 1

Introduction

This manual describes a 68HC11 based microcomputer board useful in embedded computer systems, whether in industry, Technical College, University or by the enterprising amateur.

The MPP board was originally developed for use in courses of Electrical Engineering at Ryerson Polytechnic University in Toronto, Canada. It has found its way into a wide variety of control, robotics and instrumentation projects, so we think it provides a convenient and low cost platform for embedded computing.

This manual covers material that is particularly useful getting the board customized, operational and in applied to a microprocessor control project such as a mobile robot.

Comments, requests for clarification and cash prizes are all encouraged and should be directed by email to

phiscock@ee.ryerson.ca

or by snail mail to

Peter D. Hiscocks
Syscomp Electronic Design Limited
55 Grandview Avenue
Toronto, M4K 1J1
Canada
Phone/Fax : 416-465-0325

Obtaining the Kit

A complete kit including the printed circuit board and 24 page assembly manual is available from:

Active Electronics
3790 Victoria Park Avenue
Toronto, Ontario, M5B 2K3
Canada
Phone: 416-498-9886
Fax: 416-498-9875

or Syscomp at the address given above. The EPROM and GAL in the kit are supplied pre-programmed. We also provide some email support for users of the kit.

Web Page

Late breaking information and links to the software used in the project are all available on the Ryerson WEB page:

`http://www.ee.ryerson.ca:8080/~jkoch/#mppv2`.

A Syscomp WEB page is under construction and should be available early in Y2002.

Revision History

May, 1994:	Rev 1 Design, wrote first manual
September 1994:	First release for student use.
September 1996:	Version 2 of board, revised manual extensively.
September 1997:	Added material on programming the LCD using the MPP power supply.
September 1998:	Added single chip HC11 and D-A circuits.
February 1999:	Minor corrections of parts list and schematic.
September 1999:	Added memory mapped device debugging notes.
April 2001:	Moved assembly instructions to separate document Numerous small revisions.
October 2001:	More revisions, updating

Acknowledgements

Jim Koch designed the MPP printed circuit board, maintains the WEB page and troubleshoots problems. Special thanks to Janet Fost at Active Electronics for keeping us on the rails.

Copyright

This document is ©Peter Hiscocks, 2001. If you would like to make multiple copies for student use, we would be pleased to negotiate a site license.

Chapter 2

Overview of the MPP Project

This section of the manual contains information on configuring and using the Microprocessor Project Platform¹. A monitor program and floating point package may be installed into the board EPROM and then used to develop the system software or as *software components* in the system.

2.1 Why Use the 68HC11?

Many microprocessors are suitable for embedded applications, so it is worthwhile asking why the 68HC11 is a suitable choice. Here are some of the reasons:

Many System Features In addition to the core microprocessor, the 'HC11 includes

- Serial Communications Interface (SCI) to support a serial RS232 port
- Serial Peripheral Interface, which simplifies interfacing a variety of peripheral devices
- Elaborate Timer section, with 3 input timers, 5 output timers, and an input capture counter
- 8 Channel, 8 bit resolution analogue to digital converter
- Electrically Erasable Programmable Read-Only Memory (EEPROM), useful (for example) for the storage of calibration constants in smart instrumentation

Development Support Because the processor can be operated with an external address and data bus, external memory can be attached. This allows program development to proceed under control of a monitor in EPROM with the test program in RAM, connected to the hardware of the final system. Many microprocessors require special development systems, devices which are expensive and rapidly become outdated.

Low Cost, Available The part is readily available in a variety of versions, and the cost is moderate: about \$10 CDN in single unit quantities at the time of writing.

Literature The HC11 is supported by an excellent (if somewhat dense) reference manual [1] and a data book (see for example [2]) for each variant of the basic device².

¹We thought of various fruit names for the computer, but all the good ones are taken.

²At the time of writing, these are available on request to the Motorola Literature Distribution Center, 1-800-441-2447.

Simple Architecture The design of the core microprocessor is straightforward and uses a simple memory model. (No bank switching or paging, thank you.) The interrupt system is very sophisticated for an embedded microprocessor, with hardware vectoring for the usual external interrupts and for all the timer and communication subsystems, real-time interrupt, software interrupt and illegal instruction trap. The priority of the interrupts is defined and can be changed.

Extensive Instruction Set: CISC vs RISC The 68HC11 has a very large set of machine instructions (110) and addressing modes (7) which puts it in the CISC (Complex Instruction Set Computer) camp. The other camp is RISC, for Reduced Instruction Set Computer. A popular example is the Microchip PIC series of processors.

Based on the experience of porting a substantial HC11 assembly language program to a Microchip PIC, I find that the instruction set of the 68HC11 is easier to use than the RISC instruction set. It's quite feasible to program moderate-sized programs in assembly language on the HC11. For that reason, I also believe that the 68HC11 is an easier machine for the neophyte to learn. RISC machines effectively require the use of a high level language, such as C, for anything beyond a tiny program.

However, the large instruction set of the 68HC11 adds to the size and cost of the chip and a single-chip system using the PIC is often smaller and less expensive. Ryerson EE students first learn the 68HC11 instruction set and architecture. Some students subsequently choose to use the PIC in projects, so it's possible to move from one architecture to another without a huge penalty in learning time.

Software Development Tools Available To aid software development, there are available at least two monitor programs, several assemblers, a floating point math package, a multitasking executive and various C compilers. Much of this software is in the public domain, which makes software development accessible to students and others with limited funds.

Special Embedded Application Features The processor can be operated in a single chip mode, eliminating the need for external memory parts. It is equipped with a power-down mode and a watchdog timer which can restart the program if it goes off into the weeds, because of electrical interference, for example.

Disadvantages? Well, with a 1MHz clock it's no speed demon. On the other hand, this is still something like 500×10^6 instructions per second³, lots for many embedded applications. Some of the configuration variables can only be changed within the first 64 clock cycles after reset, a minor nuisance. Finally, the BSET and BCLR (bit set and clear) instructions can only be used in page 0 (the first 256 locations) of memory.

2.2 Applications of the MPP

The MPP board is a general purpose microprocessor *system*. It has been used by students at Ryerson University to learn microprocessor technology and assembly language programming. It has been applied in a wide variety of senior variety of systems including:

- Six legged walking robot and other varieties of mobile robots
- Force feedback weigh scale
- Aerial photography camera controller

³Based on an average of two clocks per instruction

- Greenhouse controller
- The Weather Station, a student project in instrumentation
- Various power controller circuits, including high-power inverters and cycloconverters
- Slow scan television system
- Remote monitoring system for cottages
- Solar telescope controller
- Software development for single chip 68HC11 applications

Because this system has been thoroughly debugged and documented, the user should find it straightforward to assemble and program. This allows the focus of the project to be directed toward the application of microprocessor technology in a given system, rather than the details of designing and constructing a microprocessor board.

2.3 Software Development on the MPP

To begin with, consider the following scenario for software development for a microprocessor:

An engineer writes the assembly language source code on a P/C, using an editor program. She ensures that the microprocessor reset vector points at the start of the program, *assembles* the source code into a binary image file and burns it into an EPROM. Then she plugs the EPROM into its socket on the microprocessor board and presses reset. At this point, the software should execute correctly. Of course, as we all know, it will have software *bugs*, and will need to be corrected.

The fly in the ointment with this scheme is that there is no easy method of debugging the software. There are things that can be done: the program can be run on a *simulator* to test its operation, and test programs can be burned into the EPROM to narrow down problems. However, debugging is a painful process at best: each change requires reburning an EPROM. And each reburning of the EPROM may be followed by a crash of the software. As a result, this is known as the *Burn and Crash Method of Software Development*.

The simulator⁴ is a help, but there are many problems that cannot satisfactorily be resolved with a simulator.

The *Monitor Method*

The key idea here is that the program under development is loaded into RAM rather than EPROM. This has three significant advantages:

- changes do not require burning an EPROM
- the code may be *patched* in various ways to test different sections
- the machine memory may be displayed for diagnostic purposes.

⁴There are several 68HC11 simulators. One such is *The Wookie* which may be found on the Internet.

For example, *breakpoints* may be inserted in the code. The processor runs at full speed up to the breakpoint and then stops, displaying the contents of the machine registers. Or, in an extreme form of breakpointing, the machine can be made to execute one instruction at a time. The contents of machine registers and memory locations can be displayed on the host.

Notice that the microprocessor can interact with the hardware that it is reading and controlling, an essential requirement for embedded system development.

Obviously, this is a much nicer development environment than *Crash and Burn*.

To accomplish this magic, we install a small *monitor* program in the machine and point the reset vector to the start of the monitor program. When the processor is reset, it starts the monitor program.

The monitor program communicates via a serial line with an external computer, known in this context as the *host development system*, as shown in figure 2.1.

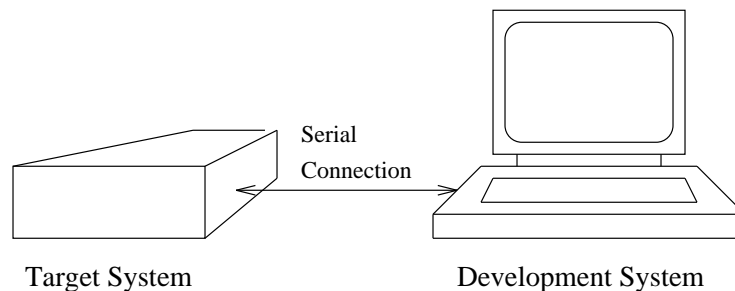


Figure 2.1: Target and Development Systems

The monitor program in the microprocessor *target* system can send the contents of the microprocessor memory for display on the screen of the host. It can also accept a program from the host and install it at a specified location in memory. Upon a command from the host, it can start a user program. Once the user program is running, it runs at full speed and has full control of the system. However, the monitor is configured in such a way that a *break* in the user program transfers control back to the monitor. The net effect is to give the host control over the execution of the test program.

In practice, test programs *crash*. That is, they don't exhibit the expected behaviour nor (if they are supposed to) do they terminate back to the monitor. In that case, the operator simply presses the reset button to regain control. Reset restarts the monitor program, which can then be used to pick through the wreckage to determine what went wrong. For example, the operator can read memory registers for clues to the crash. Or the operator can rerun a small section of the program to narrow down the source of the crash.

The monitor method is used on the MPP. A monitor program (BUFFALO) is installed in the EPROM and used to load and execute programs in RAM⁵

Once the application program is completely debugged, **then** it may be burned into EPROM. The reset vector is reprinted to the start of the application program, and thereafter it is the application program that runs when the processor is reset.

A more complete description of the monitor is given in section 5, but this will do to get us started.

There is a price to be paid for having a monitor: some of the system resources must be provided for the use of the monitor. In the case of the Buffalo monitor, it needs a small section of the microprocessor RAM for its own stack and working registers, and 8K of the memory address space for the monitor program. It also takes over

⁵An alternative monitor, *KIMSMON*, is available from the web page.

one of the machine output lines (PA3). However, once the system is debugged, it can be modified to free up these resources.

2.4 Learning to Program the 68HC11

Assembly language programming of the 68HC11 is described in a number of textbooks, references [3], [4], [5], [6].

The Reference Manual for the 68HC11 (reference [1]) and the data book for the 68HC11E1 (reference [2]) are essential reading and are available from Motorola on request. (See the Bibliography and References Section). These documents are not light reading and a beginner will need other sources of information.

You can learn a lot by looking at other people's code: the source code for the Buffalo monitor (BUFF32.ASM) makes interesting and informative reading. This and other examples are available on the WEB page.

2.5 Description of the MPP Hardware

A photo of the MPP board is shown in figure 2.2 on page 9.

A block diagram for the board is shown in figure 2.3 on page 10.

The features of the board are as follows:

Processor Motorola MC68HC11E1CFN2.

- 52 pin PLCC package
- 8 Channel, 8 bit resolution A/D
- Serial Communications Interface
- Serial Peripheral Interface
- 5 output timer channels, 3 input timer channels
- 512 bytes EEPROM
- 512 bytes RAM

EPROM The board supports one 28 pin EPROM of sizes 8K (2764), 16K (27128) or 32K (27256) bytes, selected by means of jumpers on pins 26 and 27 of the socket. The EPROM supplied with the kit is pre-programmed with a monitor program and the floating point math package.

RAM The board supports one 28 pin RAM, sizes 8K and 32K, by means of jumpers on pins 1 and 26.

Input Register One eight bit parallel input register, device 74HC244 or equivalent, is mapped into the memory address space. The input lines are connected to a header.

Output Register One eight bit parallel output latch, device 74HC273 or equivalent, is mapped into the memory address space. The output lines are connected to a header.

LCD Display A header is supplied for attachment of an LCD display. The LCD display is mapped into the memory space of the microprocessor. Notice that the LCD display attached to this port is only guaranteed to work with a 1MHz E Clock (4MHz crystal), though we've never found a problem using an 8MHz crystal (2MHz E Clock) .

Memory Map Decoder The memory map is decoded by a 16V8 GAL device, which must be programmed with the memory map. Outputs of the decoder select the EPROM, RAM, Input register, Output Register, and the LCD display header. The GAL also converts the Motorola style control lines (Enable, R/W) into Intel style control lines (Write Enable, Read Enable) for use by the various peripherals. The GAL supplied with the kit is programmed with the memory map equations.

Reset Circuit A momentary action pushbutton is provided on the board for reset.

Power Supply The board includes a 5 volt regulator with room for a small heat sink if required. The unregulated power input connection is via a power coaxial jack which mates with a 9 volt, 500 mA DC adaptor *wall wart* power supply. The unregulated power input connection includes a power diode to prevent the reverse application of power. The raw 9 volts is brought to a pad or terminal to support the powering of single supply op amps from a supply greater than +5 volts. Bipolar power of +/-10 volts and a few milliamps is available from the RS232 level converter. This might be useful to power one or two op amps or other low current devices that require bipolar power.

Wire Wrap Area Unused board area is given to a matrix of holes on 0.1 inch centres for application-specific circuitry.

Bus Expansion Header This connector may be used to attach memory mapped peripherals. For example, video memory could be mapped into the address space using this connector and the Select0 address decode signal. (An example is shown in section 3.8). The signals that are brought to the Bus Expansion Header are shown in figure 3.4 (page 30). Spare pins can be jumpered to signals (such as IRQ) to suit the project.

The **Select0** signal is an output of the GAL memory map decoder. This select signal replaces the upper address lines A12 through A15.

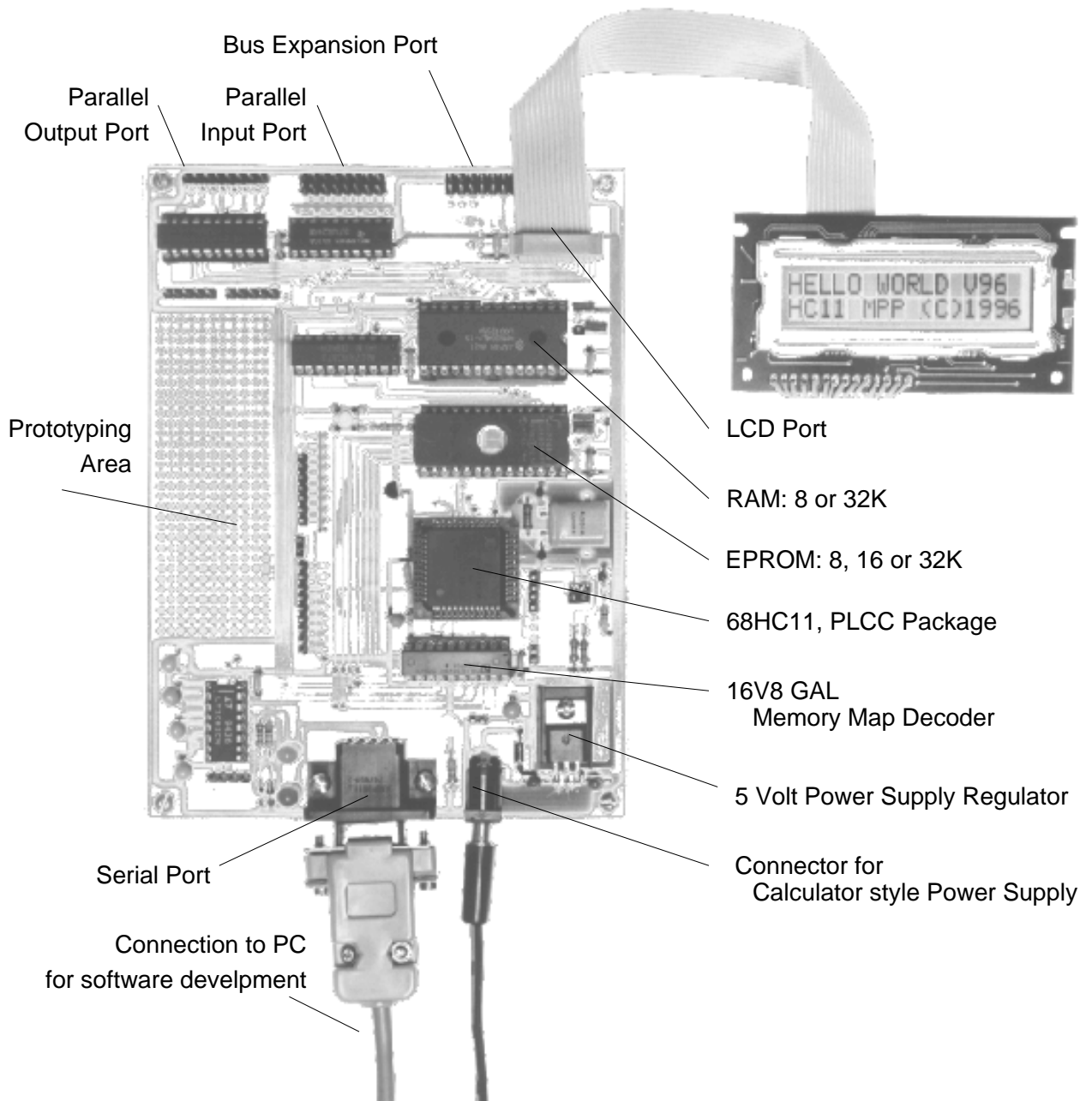
The **Address Strobe** (AS) line is brought to an edge connector pin so that the address and data may be demultiplexed on a remote board to minimize backplane wiring in a multi-card system.

The +5 volt and power ground lines are available on the expansion connector.

Crystal Either a 4MHz or 8MHz crystal may be used, but the LCD port may be used only with a 4MHz crystal. Unless you have good reason to use an 8MHz crystal, 4MHz is advised.

Serial Port PCB mount DB-9 connector on the board is configured for a direct serial (COM) port connection to a PC. (No crossovers or null modem are required.)

A Maxim MAX232 or Linear Technology LT1081 translates the 5 volt logic signals to and from the RS-232 logic swing and helps isolate the microprocessor board from the P/C development system.



/books/hespec/figures/mpp-board-image.fig

Figure 2.2: MPP Board

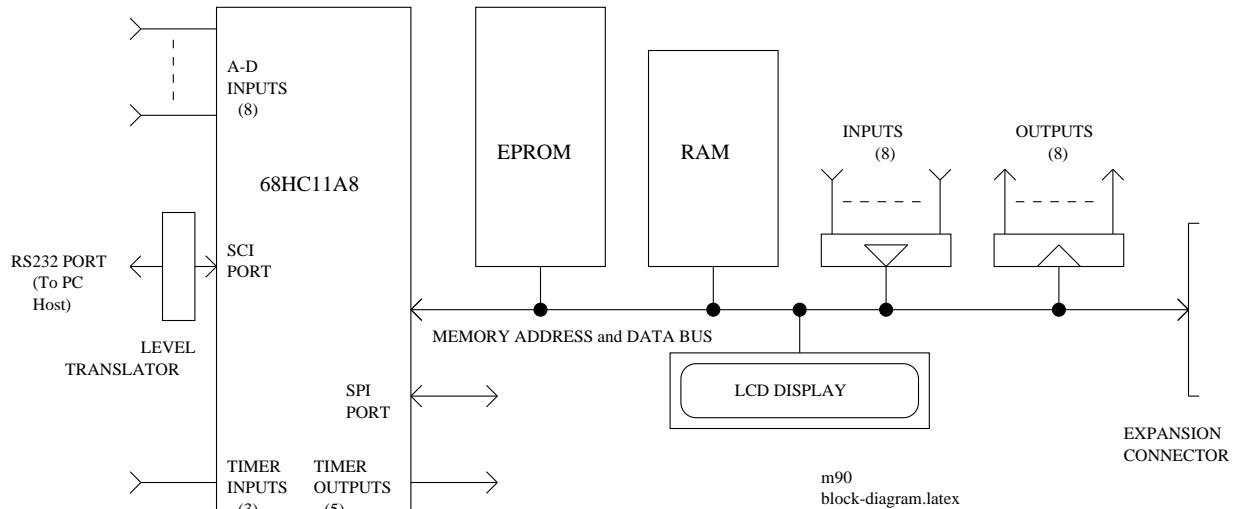


Figure 2.3: MPP Block Diagram

Synchronous Port Interface The SPI is one of several ways to connect external devices to the microprocessor, either to sense external signals or generate signals to control other devices. The following signals are brought out to a header to facilitate use of the SPI.

- MISO (PD2)
- MOSI (PD3)
- SCK (PD4)
- SS* (PD5)

. An application of the SPI to add 8 bit input and output ports is shown in section 3.6.

Pullups on 68HC11 Pins The following pins are pulled up by resistors to +5 volts: XIRQ, IRQ, MODA, MODB, IC1, IC2, IC3. Each pin has an associated wire wrap terminal to which signals may be connected.

A-D Inputs Inputs to the A-D, VRH and VRL are brought out to wire wrap pins. VRH is not connected to +5 volts, and VRL is not connected to ground, since they should be connected to the analog section of the project. For non-critical A-D applications, the VRH and VRL pins may be jumpered to +5 volts and ground.

LED Indicators Three LEDs are provided to indicate the board status and help diagnose problems:

- Power
- Serial port transmit
- Serial port receive

Single Stepping In order to support single stepping operation using the Buffalo monitor, OC5 must be connected to XIRQ. Two adjacent header pins are provided for this purpose: jumpering them connects OC5 to XIRQ.

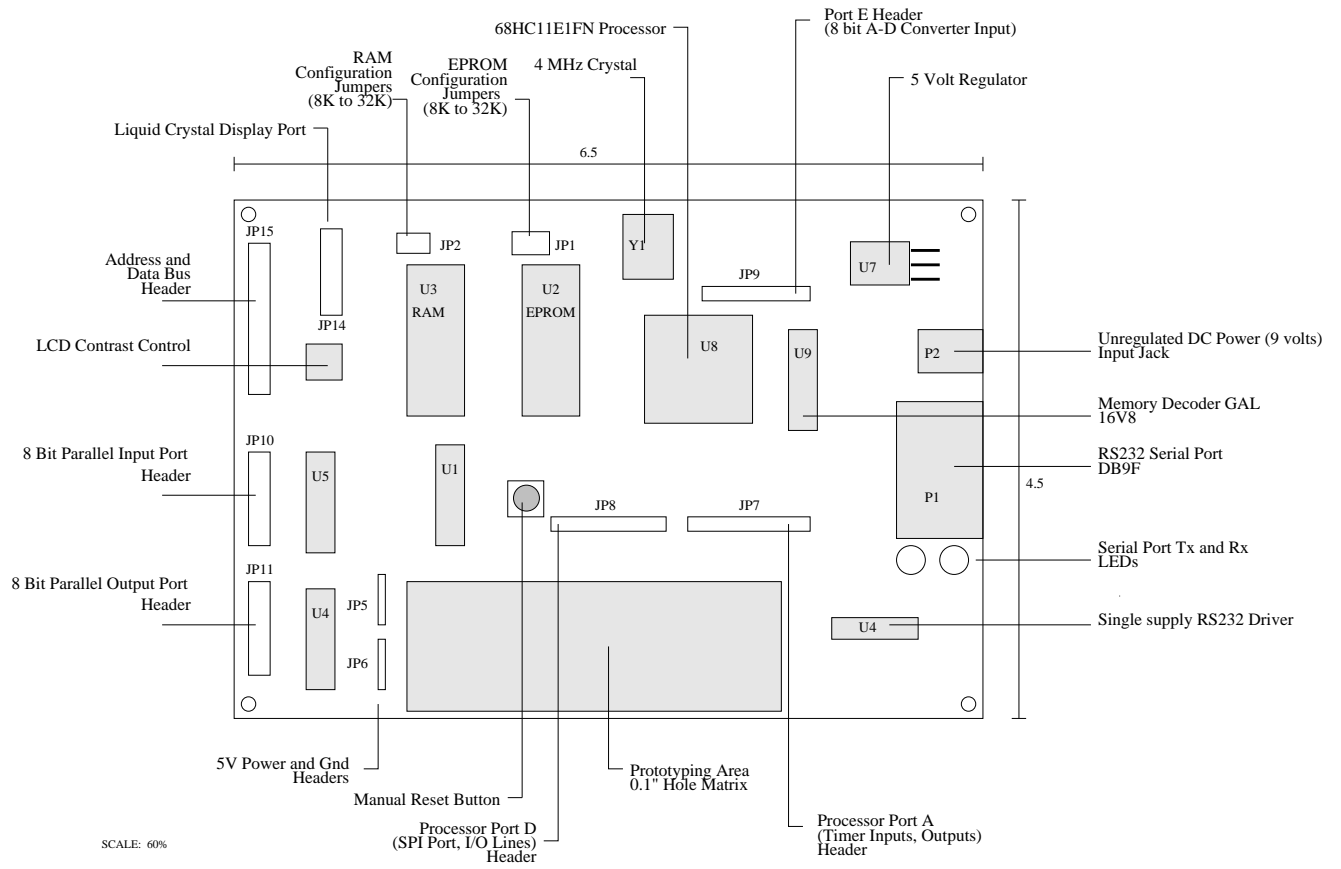


Figure 2.4: Jumper Positions and Functions

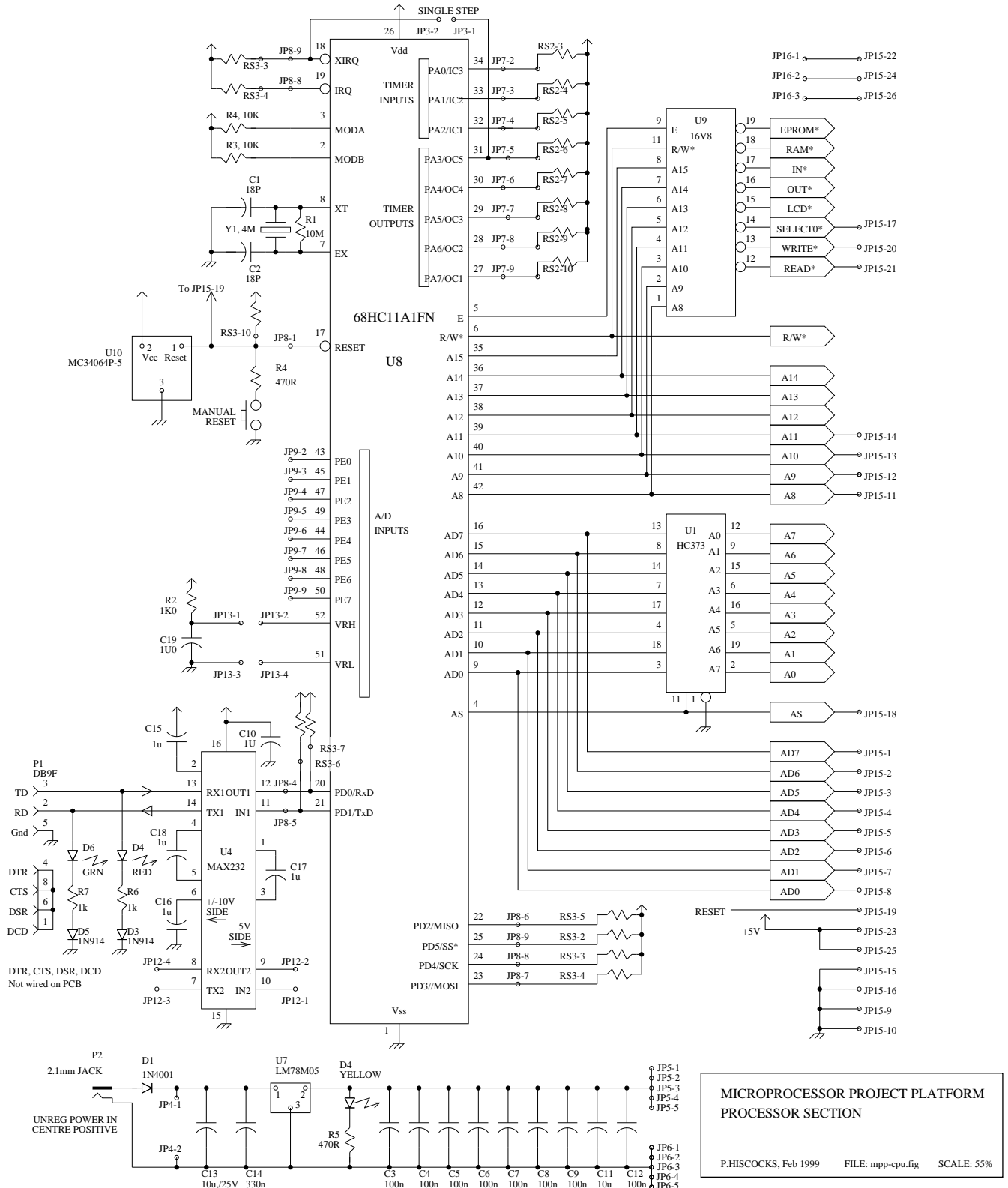


Figure 2.5: MPP Schematic, Processor Section

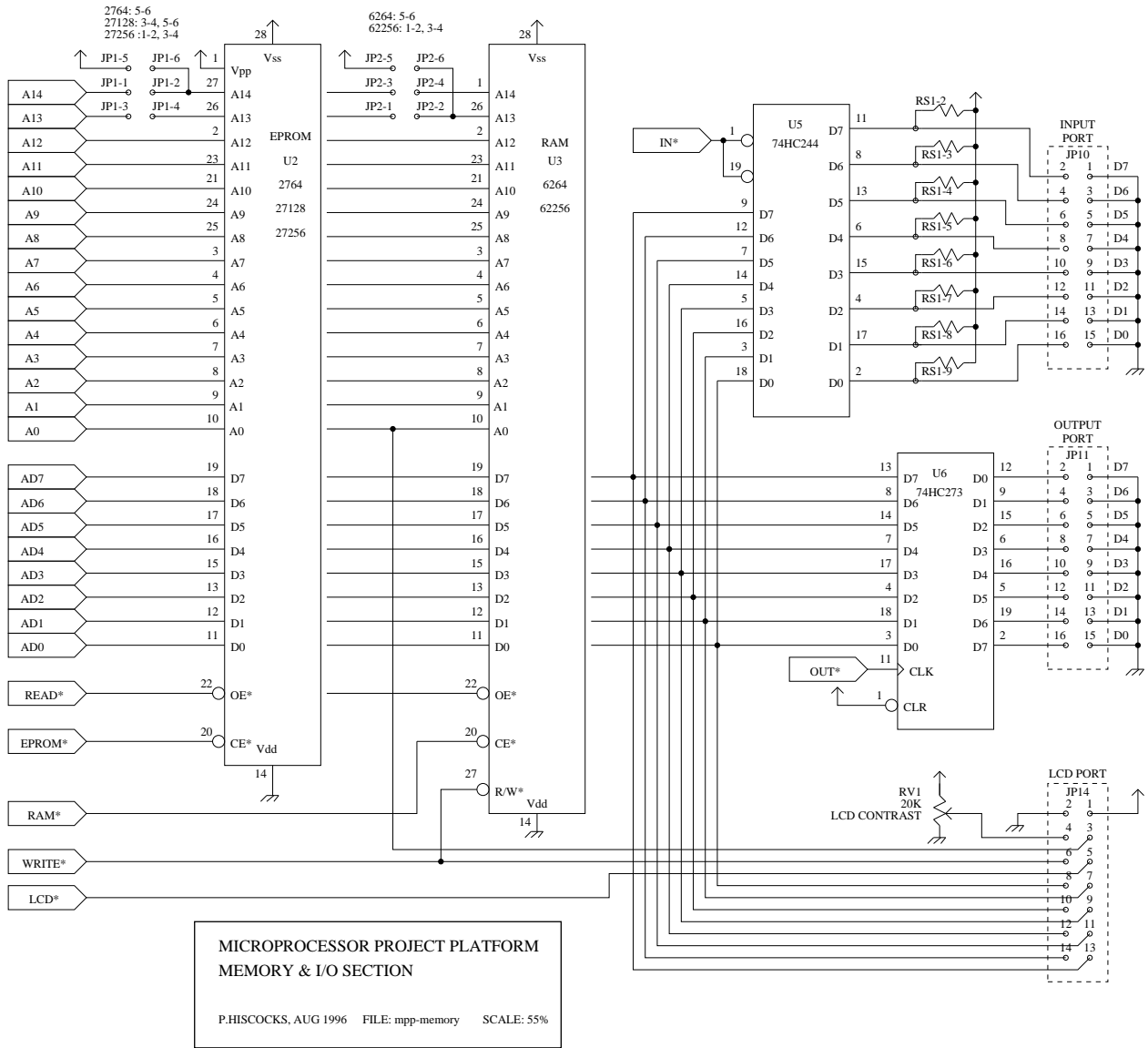


Figure 2.6: MPP Schematic, Memory Section

2.6 MPP Default Memory Map

The *memory map* for a computer system shows the addresses of the various hardware devices and resident software. The default memory map for the MPP system is shown in figure 2.7.

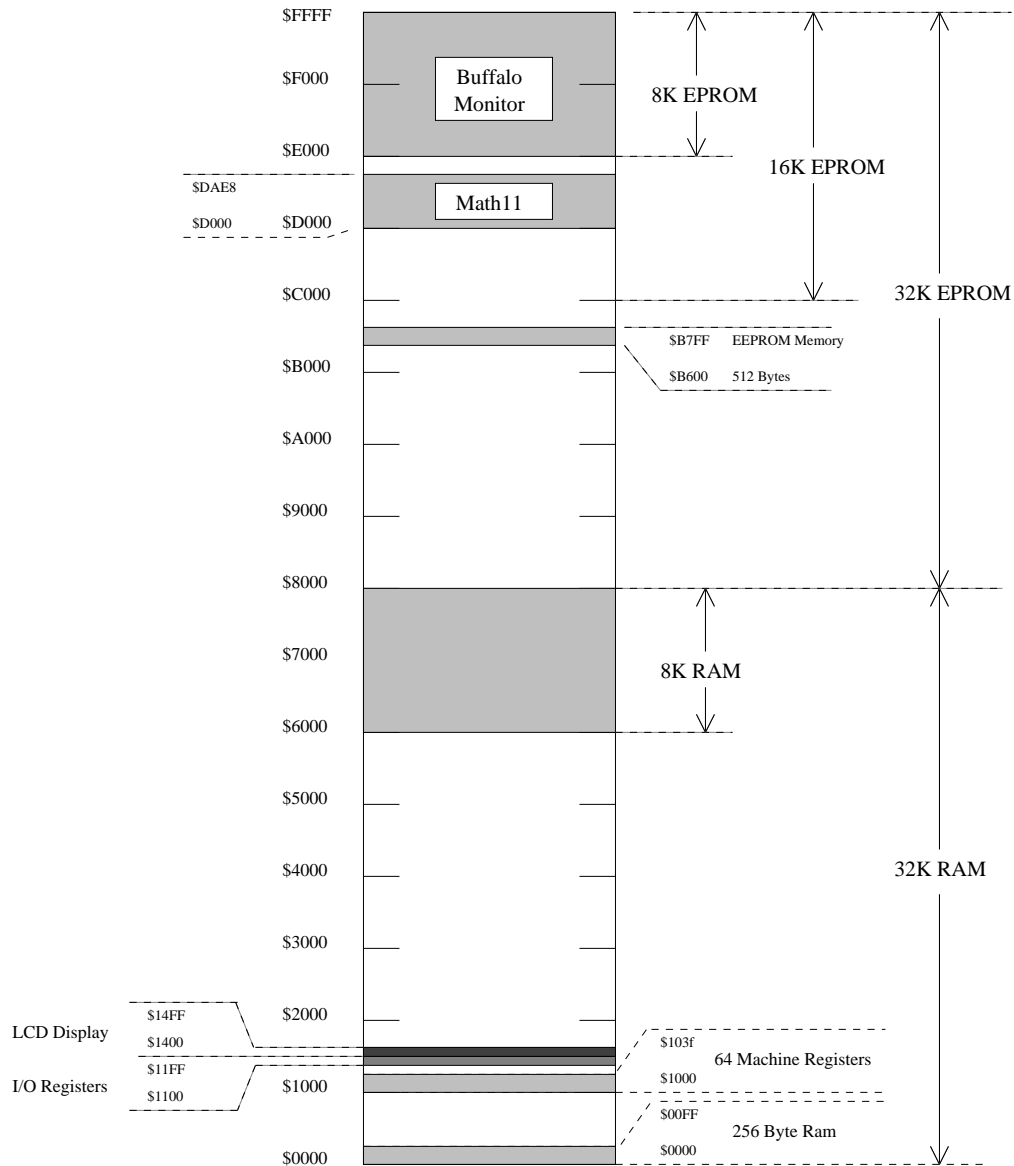


Figure 2.7: MPP Memory Map

The following devices are memory mapped:

EPROM The EPROM may have three different sizes: 8k,16k or 32k bytes. In all cases, the end of the EPROM must occur at the top of memory in order to store the processor reset vectors. Thus the starting address of an EPROM depends on its size.

The default memory map decoder enables the EPROM select line for the upper half of memory, so that any sized EPROM will work without changes to the GAL. Of course, if your program reads an address at which there is no EPROM, it will read garbage. So the EPROM jumpers need to be changed for different sizes of EPROM, but the GAL does not.

If, however, you wanted to use an 8K EPROM and map some other device into the unused space in upper memory, you would have to change the GAL equations.

Notice that a 32K EPROM overlaps the processor internal EEPROM between \$B600 and \$B7FF (unless the internal EEPROM is disabled) in which case EPROM addresses between \$B600 and \$B7FF are not useable. (The 68HC11 gives priority to its internal devices when they overlap with external devices in the memory map.) If this is a problem, the EEPROM can be disabled.

The EPROM supplied with the MPP board kit is 16k bytes in size in order to store the 8k Buffalo Monitor and 3k MATH11 routines.

RAM The static RAM may have two different sizes, 8K or 32K bytes. In order to be compatible with the three different sizes of EPROMs, the RAM is located in the bottom half of memory. A 32K byte RAM overlaps a variety of things in the bottom half of memory, and so is effectively not useable below address \$1800. Thus the memory decoder is programmed to enable a 32K RAM chip only between addresses \$1800 and \$7FFF, so part of the 32K RAM chip is unuseable. An 8K or 32K RAM chip may be used without changes to the decoder GAL, but there must be valid memory at whatever address is read from or written to: there is no hardware to prevent reads and writes to addresses that do not have RAM⁶. When using a 32K RAM chip, avoid locations below \$1800.

LCD Display The base address of the LCD display is established at \$1400. With this base address, the LCD control register is at \$1400 and the data register at \$1401. Because of incomplete address decoding, the display occurs throughout the address block \$1400 to \$14FF *address aliases*. (Another way to indicate the base address of the LCD is as \$14xx, where *xx* indicates *don't care*).

Input-Output Registers These are both located at a base address of \$1100. A *write* directs the data to the output register: a *read* gates the input register onto the data bus. As in the case of the LCD display, both registers are aliased throughout addresses \$1100 to \$11FF. (Also see the note in section 2.6.1 below.)

Select 0 One additional spare device select line is available from the GAL decoder circuit and brought out to the bus expansion connector. In the default design, this select line is disabled, so no space is allotted for them on the default memory map. If you need this control line to add something to the memory map, check out the information on modifying the memory map in section 7.7 and section 7.3.

Also shown on figure 2.7 are the map locations of the processor internal registers (\$1000 to \$103F) and internal memory (\$0000 to \$00FF). As in the case of the EEPROM mentioned above, any external devices mapped to these addresses would be ignored.

⁶Many computer systems have hardware support to signal when an illegal memory address is accessed. How could that be done here? Well, you could program the GAL to generate an active low signal whenever an illegal address is accessed, and the signal could be used to actuate an interrupt line, XIRQ for example. The interrupt service routine would print an error message, together with the offending instruction and address. A good student project, by the way.

2.6.1 The ROM Error Message

When you use the monitor program MM (Memory Modify) command to change a byte in memory, you may get the ROM error message. This is the monitor program saying in its own inimitably brief way: "I wrote something to memory but couldn't read the same thing back". The monitor then assumes that the memory area in question is read-only.

However, many areas of memory, especially machine registers and output ports, are legitimate to write to and do not read back the same data. For example, when you write to the Output Register at \$1100, you do indeed write to the hardware. However, the monitor reads back contents of the Input Register, and these are usually different than what was just written. So the monitor generates the ROM error message but it may be ignored.

2.7 The MPP Power Supply

The 'standard' board power supply is an AC adaptor power supply of 9 volts and output current capability of 500 mA maximum. This is then regulated by a version of the 7805 type integrated circuit three terminal regulator (the 78M05CT) that has a current limit of 500 mA. Under normal conditions, the MPP should require no more than 100mA or so, which is well below the capabilities of the AC adaptor transformer and regulator IC. (The processor is specified requiring 25 mA under normal operation.) The power supply specified in the parts list should be sufficient for most student requirements. If a more robust power supply is needed, replace the regulator with a standard 7805 device (1 amp output current limit) and the AC adaptor with a transformer-rectifier-capacitor-filter capable of at least 7 volts and 1 amp current output. In this case, a larger heat sink on the regulator IC will be required.

We have seen some attempts to use a 6 volt adaptor to power the MPP board. This is very marginal and will result in mysterious resets to the board, so it's not recommended.

On the other hand, a 12 volt adaptor will work under normal circumstances. The heat sink is noticeably warmer, however, and the amount of additional current that can be supplied to auxiliary circuits is consequently much less.

If you are using the MPP board in a mobile robot application, you must take extreme care to decouple the motor power supply from the microprocessor supply. It is possible to use the same battery for both, but the motor system should have its own regulator and LC filter. Otherwise, noise spikes and load transients will cause misoperation of the microprocessor board.

2.8 Software Distribution

The files shown below may be accessed via the Ryerson Electrical Department WEB page (see Introduction), which users are welcome to download.

README A description of the documentation in the distribution, similar to this section, but including last minute changes. (Some of the file names in this section may not be exactly correct: check the README file).

Editor, QEDIT.EXE This is a simple editor that runs on a DOS system. Students may prefer to use their own editor such as NOTEPAD on a Windows95 system.

GAL Configuration File, DECODER.ASC This file is the programming file for the memory map GAL chip.

GAL Documentation File, DECODER.DOC Pinout and other documentation for the memory map decoder GAL chip.

GAL Compiler, `PLAN11` or `OPAL` This program (from National Semiconductor) converts the GAL programming file (eg, `DECODER.ASC`) to the JEDEC form that can be used by the GAL programmer (eg, `DECODER.JED`).

Cross Assembler, `AS11.EXE` This program runs on a DOS computer and accepts an ASCII text file of 68HC11 source code and generates the `.S19` and `.PRN` versions of the file. The `.S19` file may be downloaded from the P/C into the MPP.

There is also a Unix version of the assembler that runs very nicely under Linux. For a variety of reasons, the Linux environment is much more convenient than the Windows environment to use for program development.

Assembler Manual, `ASREF.MAN` Operating manual for Motorola Freeware assemblers, including `AS11.EXE`.

Terminal Emulator, `PROCOMM.EXE` This program converts the DOS host P/C into a dumb terminal for communication with the MPP. Students may prefer to use a different terminal emulator, such as the Windows95 `Hyperterminal`.

Terminal Emulator Documentation, `PROCOMM.MAN` A short summary of commands and the use of `PROCOMM` for cross-development with the MPP. The complete documentation file for `PROCOMM`, `PROCOMM.DOC` is somewhat lengthy, but is also provided for reference.

Monitor Program, `BUFF32.ASM` This program is assembled and burned into the MPP EPROM to provide an operating system on the MPP. The monitor commands and idiosyncracies of the monitor are described in chapter 5. We have slightly modified the source code to better suit this application, and it is known as `BUF32MOD.ASM`.

Floating Point Package, `FP11.ASM` Source code in 68HC11 assembly language for a floating point package for the 68HC11.

Floating Point Package Documentation, `FP11.DOC` Brief description of the routines and registers used in the floating point package. Section 6.6 of this manual describes in detail how to use `FP11`.

Multitasking Executive, `MCX11.ASM` Source code for a multitasking kernel on the 68HC11.

Multitasking Documentation, `MCX11.DOC` Manual on the use of the multitasking kernel.

Library of Subroutines A variety of useful subroutines are stored in this sub-directory. Students are encouraged to use these routines in their programs. If you develop a useful subroutine, please email it to the author for inclusion in the library.

This software runs under DOS or in a DOS window under the Microsoft Windows operating system. The minimum environment for cross-development of MPP programs is a P/C XT machine, so it should be possible to scrounge something suitable.

2.9 Setting Up Communications

Assuming that the MPP is working, it should now be possible to establish contact with the Buffalo monitor. Before we launch into the nitty-gritty of making this work, a bit of explanation is in order.

2.9.1 The Dumb Terminal

A *terminal emulation* program is a program that sucks the intelligence out of a personal computer and turns it into a relic of the past: the *dumb terminal*. To appreciate the absence of functions of the classic dumb terminal, we have to journey back to the Jurassic Age of computer technology, the Time of the Mainframe, between perhaps 1960 and 1975. Mainframe computers were large, expensive, and had impressive control panels that only a selected few priests were allowed to operate. The dinosaurs required air conditioning and raised floors (to accommodate cables) and so were kept in separate rooms. Lesser mortals accessed programs via the *glass teletype*, a terminal equipped with a video display and a keyboard, the dumb terminal. IBM had its own standard for these things, and a few of them can still be found in the corridors of high technology enterprises, line cord removed, evidence that yesterdays status symbol is tomorrows computer trash.

The dumb terminal existed at a time when computers and electronic devices were expensive. From an economic point of view, it made sense to have one hideously expensive mainframe and a bunch of low cost terminals to communicate with the mainframe. Each terminal had the absolute minimum of intelligence to detect a key press on the keyboard and spit that out a connector on the back, or to receive a character stream and display that on the video screen. The electronics was 'bubblegum' LSI logic. Later models might have had a UART or primitive microprocessor.

Even though mainframes and dumb terminals are no longer with us, the standard lives on as the lowest common denominator for communication with a computer. Thus, in the situation that concerns us, the MPP will communicate with a device much like a dumb terminal. Now dumb terminals are easily had surplus these days for about \$10, but it's rather a clunky and useless thing to keep around. Instead, we load a program into our P/C's that *emulates* a dumb terminal. As a bonus, we can still use all the nice P/C features, such as the hard disk for storage and other utilities, such as the 68HC11 assembler.

(Incidentally, any computer running a terminal emulator will work equally well. I used an Atari ST for this purpose for many years. However, most development software runs only on as DOS P/C.)

Most dumb terminals communicated with a mainframe via a serial RS232 line, a standard that continues to be widely used as the COM port of a P/C. In its minimum form, the serial communication link consists of three lines: Tx (transmit), Rx (receive) and Ground. Other pins are used for flow control and more obscure functions that only telephone engineers care about. We shall use the three wire connection.

Getting the dumb terminal to talk to the MPP requires sorting out two main issues - getting the Rx, Tx leads connected properly, and establishing the correct protocol.

2.9.2 Communication: Step by Step

1. Find a terminal emulator program and install it on your computer. A typical terminal emulator for DOS machines is `PROCOMM.EXE`, which is available as part of the MPP software distribution.
2. The terminal emulator will have a means of setting the speed and format of the characters it sends and received. Configure the terminal emulator for 4800 baud, 8 data bits, 1 stop bit, no parity. (This may not be the correct speed, but it will do for now.)
3. Identify the COM port on the back of your computer. This can be a bit tricky: the `PRINTER` port is often a similar type of connector. Serial ports tend to be male and printer ports female, but this is not a fixed rule.

The next step is to identify the Tx, Rx and Ground pins on this connector. If you have a manual, this will help. Unfortunately, Tx is sometimes considered to be *from the computer* and other times *to the computer*.

For a 25 pin connector, the relevant pins are 2,3 and 7. Ground is always on pin 7. (You need bright light and good eyesight to read the numbers of the pins, but they are there. Small grabbers of the type used to connect to IC pins are handy in clipping to the pins of the connectors.)

One of pins 2 and 3 will be sitting at -10 volts or so. This is the transmit pin. The other pin will be sitting at 0 volts. This is the receive pin. With the terminal emulator program running, watch the transmit pin on an oscilloscope and press a key on the keyboard, and you should see a train of pulses to +10 volts.

If the COM port is a 9 pin connector, the relevant pins are 2,3 and 5, with ground on pin 5.

Incidentally, if you are using COM1 for a mouse or modem, you can probably configure the terminal emulator to talk to COM2. If you don't get any activity on the COM port pins when you type characters, try the other COM port.

If you expect to do this very often, a handy commercial gizmo is the COM port tester. This consists of two DB-25 connectors with LEDs on all the important pins. Simply plug it into a DB-25 serial port and the LED on the Tx line will flash. If you have two machines talking to each other over a serial line, you can see the activity on both Tx and Rx LEDs.

Another handy device allows you to jumper various pins of the COM port together, to facilitate swapping pins 2 and 3 for example. Each of these devices is about \$10.

4. Having identified the transmit, receive and ground pins on the computer COM port, wire up a cable with a suitable connector for the computer end and a DB-9 connector to plug into the MPP serial port. It helps to use 24AWG or finer stranded wire (do **not** use solid wire) and clamp the cable so it is strain relieved at both connectors. If you expect to use a different machine (for example, the machine in one of the Ryerson labs), make up a suitable cable for that as well. Incidentally, should you lose the information for the MPP serial port connections, the same technique may be used there to identify Tx and Rx.
5. We are now ready to communicate with the MPP. Connect the computer COM port to the serial port of the MPP such that Tx on the host computer is connected to Rx on the MPP and vice versa, and so that the two ground pins are connected. Power up the MPP and the Buffalo signon message should appear on the terminal emulator screen.
If garbage appears, probably the baud rate is wrong. Try changing the host baud rate to 2400, 4800 and 9600. One of those should work⁷.
6. When you get the monitor signon message, try various monitor commands. For example, MD should do a Memory Dump of a section of MPP memory.
7. Finally: Congratulate yourself. You have a working 68HC11 Development System.

2.10 A First Program on the MPP

This section shows the development of a *very* simple program for the MPP. You can extrapolate to something more weighty, but this program simply adds the numbers 3 and 5. Hopefully, the computer will agree that the answer should be 8.

We assume in this section that you've had some experience programming in machine language. If not, you might wish to look at one of the references. We also assume that you're working in the DOS environment.

⁷When you change the baud rate in Hyperterminal, you have to *disconnect* and then *connect* to have it take effect. Otherwise Hyperterminal lies: it tells you a different baud rate from the hardware setting.

2.10.1 Step 1: Write the program

Using your favorite text editor, (QEDIT under DOS, Clipboard under Windows: whatever), write the program. It should look something like this:

```
* Program to add two numbers
  ORG $6000           Starting address of the program
  LDAA #3            Get the first number
  ADDA #5            Add the second number
  NOP                No operation: We'll explain this one later
```

(Notice that all instructions and assembly directives must not appear in the first column. Only symbolic address labels may appear in the first column.)

Save the program as MYPROG.ASM. (The 'ASM' suffix indicates that this is an assembly language source code file. There is nothing magical about 'ASM': you could use 'SRC' if you prefer.)

2.10.2 Step 2: Assemble the program

In the DOS environment, call the assembler with the incantation

```
as11 myprog.asm -l cre >myprog.prn
```

(If you get tired of typing this in, create a batch file containing that line.) This should have the effect of starting up the assembler and very shortly thereafter, two new files should appear on your hard drive:

```
myprog.s19
myprog.prn
```

You can load either of these into your text editor to examine. The 'prn' file is a *listing* file (some people use the suffix *lst*). The first column contains the line number. The second column contains the address of each instruction. The third column is the machine code for the instruction. The rest of the line is the original assembly language text.

```
Freeware assembler ASxx.EXE Ver 1.03.
0001                * Program to add two numbers
0002 6000           ORG $6000
0003 6000 86 03    LDAA #3
0004 6002 8b 05    ADDA #5
0005 6004 01      NOP
```

Number of errors 0

If there are any errors, they are documented here and you have to go back to your source code file and fix them before proceeding. This listing file is absolutely essential information during debugging: it shows where different components of the program are located in memory.

The myprog.s19 file is an encoded version of your program (an *S-Record*) which can be uploaded into the MPP board. It looks like this:

```
S108600086038B05017D
S9030000FC
```

Reference [6] has a description of the format of the S19 file. It's not important at this stage, but understanding it can be useful in some situations⁸.

2.10.3 Step 3: Upload the S19 file into the MPP board

Presumably, you have established contact with the MPP board, using a terminal emulator. Start up your emulator program. (Procomm under DOS, Hyperterminal under Windows95, or Seyon under Linux, for example. Incidentally, this is where a multi-windowing GUI is really convenient: you can start up your editor with source code in one window with the terminal emulator in a second window.)

Establish contact with the board. For example, press the reset button and the terminal should show the signon message.

Enter the 'load' command: `LOAD T`. The MPP board is now waiting for an S-Record to be uploaded.

Choose the 'upload' option for your terminal emulator, enter the file name (`myprog.s19`), and the file should upload into the MPP board.

2.10.4 Step 4: Check that the program loaded correctly

Do you trust your computer? You shouldn't! Check that the file loaded into the MPP memory correctly. To do this, do a memory dump of addresses starting with location \$6000, the start of your program, with the command `MD 6000`. You should see the sequence of bytes:

```
86 03 8b 05 01
```

This corresponds to the information on the .prn file and confirms that the program is indeed in memory.

2.10.5 Step 5: Test the program

We wish to run the program starting at address \$6000. The last instruction is a NOP at address \$6004, where we want to exit the test program and break back to the monitor. The first thing to do, then, is to establish a break point at address \$6004, with a monitor instruction something like

```
br 6000
```

(Just to make it confusing, the monitor assumes all numbers are hexadecimal notation: you don't need the dollars sign.)

Now you can run the program:

```
go 6000
```

When the program completes, a few microseconds later in this case, it breaks back to the monitor, displaying the register contents. It should show that the 'A' accumulator contains the sum of 5 and 3.

That's it! Program development occurs with the same cycle: edit, assemble, upload, test. Oh yes, and **debug**.

Once you can write small routines, and make them work, you should begin to think about *designing* the code. Like any other engineering design skill, there are good and bad ways to design a computer program, and there are techniques that will overcome some of the potential problems. Designing and building a large program is quantitatively and *qualitatively* different from writing little programs, and requires careful thought.

⁸For example, it's possible to combine S19 files using an editor, if you understand the format.

Bibliography

- [1] Motorola 68HC11 Reference Manual
Motorola Publication M68HC11RM/AD, Rev 3
General applications information on the 68HC11
family of microprocessors. A 'must have'.
Available on request from the Motorola
Literature Centre, 1-800-441-2447

- [2] Motorola 68HC11 E Series
Motorola Publication M68HC11E/D, Rev2.0
Contains data on the E series
of devices, including electrical characteristics.
Available on request from the Motorola
Literature Centre, 1-800-441-2447

- [3] MC68HC11: An Introduction: Software and Hardware Interfacing
2nd Edition
Han-Way Huang
West Publishing Company, 2001

- [4] Microcontroller Technology: The 68HC11
2nd Edition
Peter Spasov
Prentice Hall, 1996

- [5] The 68HC11 Microcontroller
Joseph D. Greenfield
Saunders College Publishing, 1992

- [6] Data Acquisition and Process Control with the 68HC11 Microcontroller
Fredrick F. Driscoll, Robert F. Coughlin and Robert S. Villanucci
Macmillan Publishing Company, 1994

- [7] Using the 68HC11 Microcontroller
John C. Skroder
Prentice Hall, 1997

- [8] The 68HC11 Microcontroller
Michael Kheir
Prentice Hall, 1997

- [9] Programmable Logic Devices Data Book and Design Guide
National Semiconductor Publication 400081, 1993
Note: If you have an earlier version of this data book, hang on to it. This latest version does not contain diagrams of the various PAL configurations, and the earlier ones do.

- [10] Programmable Logic Devices
Robert G. Brown
Electronics Now, May 1994, pp31-34,38,84

- [11] Build This PLD Programmer
Robert G. Brown
Electronics Now, May 1994, pp39-44, 84

- [12] Digital Design: Principles and Practices
John F. Wakerly
Prentice Hall, 1994

- [13] Digital Logic Design
John P. Hayes
Addison Wesley, 1993

- [14] ROM Monitor Tips and Tricks Aid Debugging Effort
Peter Lawson and Andy Lanz
EDN Software Engineering Special Supplement
(issue unknown)

- [15] Sweet-16 Revisited
Charles F. Taylor
Micro, the 6502/6809 Journal, January 1982, pp 25-42
A 16 bit virtual architecture based on an 8 bit microprocessor.

- [16] Musical Applications of Microprocessors
Chapter 18, Musical Synthesis Software
Hal Chamberlin
Hayden Book Company, 1980
A double precision math package for the 6502 micro.

- [17] Structured Programming in 6502 Assembly Language
Kim G. Woodward
MICRO-the 6502-6809 Journal, August 1982, pp 43-46
Decision structures in 6502 assembly language

- [18] Philips Components
1990-91 Data Book
Philips Passive Components Group
2001 W. Blue Heron Blvd.
P.O.Box 10330
Riviera Beach, Fl. 33404, USA
- [19] Operational Amplifier Circuits, Theory and Applications
E.J.Kennedy
Holt, Rinehart and Winston, 1988, pp 490-497