A Development Aid for Microprogrammable Microprocessors

Abstract

The aim of the project is to provide a development tool which enables programmes and microprogrammes, written previously, to be evaluated on the hardware of microprogrammable bit slice microprocessors. The system is based on a PDP11 minicomputer with a broad, programmable interface controlled by a Motorola MC6800 microprocessor. Simulation of the bit slice microprocessor's main memory, microprogramme memory, clock etc. is provided by the interface or by the PDP11 through the interface. It is attempted to reproduce normal operating conditions of the bit slice devices as closely as possible.

The electrical interface consists of two 64 bit ports, one for outputs and the other for inputs. The outputs are tri-state, i.e. low potential, high potential, or high impedance (virtually disconnected), and enabled in blocks of eight bits. Latches (registers) are used for inputs in order that transient outputs from the microprogrammable devices may be stored. Again these are clocked in blocks of eight bits. It is quite permissible to connect the outputs and inputs together, however, care must be taken never to connect together enabled outputs of differing logic level. A bidirectional data bus is formed in this manner, the read/$\overline{\text{write}}$ line controlling which outputs are enabled.

Mechanically the interface is built on three boards and plugged into a mother board. From these come flying leads carrying data and control lines to a block with two edge connectors, which are also supplied with +5V. The microprogrammable microprocessor, sequencer units etc. would be assembled on two cards and plugged into the edge connectors.

In normal operation, all control of the interface would be implemented either by hard wiring on the bit slice processor cards or by software generated on the PDP11. Four sets of information are required to define the software, viz. details of the connections to the interface, dumps of the main and microprogramme memories, and details of what monitoring facilities are required. Using this data software both for the PDP11 and for the MC6800 is written, the latter being loaded into the M6800's memory from the PDP 11.

The design, construction, and testing of the interface is described. Software for the system depends on the bit slice devices chosen; the approach to software design is discussed with illustrations from two very different devices. An example of the use of common routines developed for communication between the PDP11 and the interface is given.

David Salmon,
University of St. Andrews.

AIEN APIΣTEYEIN

A Development Aid for Microprogrammable Microprocessors

David Salmon, B. Sc.(Physics),
Dept. of Computational Science,
University of St. Andrews.

DECLARATION

I hereby declare that this thesis has been composed by myself; that the work of which it is a record has been done by myself; and that it has not been accepted in any previous application for any higher degree. This research concerning A Development Aid for Microprogrammable Microprocessors was undertaken on 1st October 1976, the date of my admission under ordinance 51 for the degree of master of science M. Sc.

David Salmon

Contents

CHAPTER 1

Introduction

The advent of the bit slice microprogrammable microprocessor has given a new impetus to the design of compact, yet powerful computers. It is now a practical proposition for two or three people to design and build a viable machine. If bit slice microprocessors are being investigated, it is likely that several designs will be considered, either based on different structures or on different components. Much is common to all resulting designs of computer, so a versatile development tool has been created for use with all prototype systems. This development aid both provides common facilities such as memory, and incorporates diagnostic aids that would not form part of a final, self-contained computer.

1.1 Bit Slice Microprocessors

Electronic logic circuits may be classified into two broad categories depending on the dominant method of indicating logic levels - voltage or current. These are, of course, related, but some circuits require the passage of a continuous current (within certain voltage limits) in one direction or the other through the input terminals to establish a logic value, while others draw current only during transients or due to imperfections. MOS technology is voltage operated. The circuits are not designed to pass large currents so power dissipation is low and thus the elements of the circuits can be small and simple. In every circuit there is a certain amount of stray capacitance and this, coupled with the low currents, means that there is a significant time taken to change from one logic level to another ($\frac{dV}{dt} = \frac{I}{C}$). Bipolar technology is current operated and the currents used to charge stray capacitances are insignificant compared with the signalling currents. The elements of the circuit have to be larger than MOS in order to handle the current and power, so it is not possible to fit so many into a given area of integrated circuit. In general it can be said that the speed of operation times the power dissipated for a given operation is similar in all technologies, and all advanced integrated circuits dissipate the same amount of power.

Most microprocessors, including the Motorola MC6800, are miniature versions of a conventional computer's central processor. They have a predetermined word length, predetermined instruction set and a predetermined size and format of address space. Due to the complexity of their logic they require to use MOS technology thus a few microseconds are required for the simplest functions on eight bit operands.

With bit slice devices the logic of the processor is divided into its elements (arithmetic and logic unit plus various controllers) which are in turn sliced into units, across the word, of a few bits. These slices are implemented using bipolar integrated circuits. Bit slice integrated circuits may be wired together in order to manufacture a computer of any reasonable word size and architecture. The computer is microprogrammed, hence the instruction set may be tailored to suit almost any language or application.

The essential elements of a bit slice processor based computer are the calculating unit, microprogramme memory, main memory and various control units including the generator for the microprogramme address. Figure 1 illustrates how a typical system might be configured. The blocks represent:

i Four by four bit arithmetic and logic units, with a carry look-ahead unit to increase the speed of arithmetic operations.

ii Two by four by four array of 512 by 8 bit read only memories for use as a microprogramme store.

iii Three by four bit sequencer elements.

iv Main memory.

Neglecting certain gates and buffers that are not shown, and the main memory array, it is seen that it is possible to build a 16 bit computer from well under 50 integrated circuits.

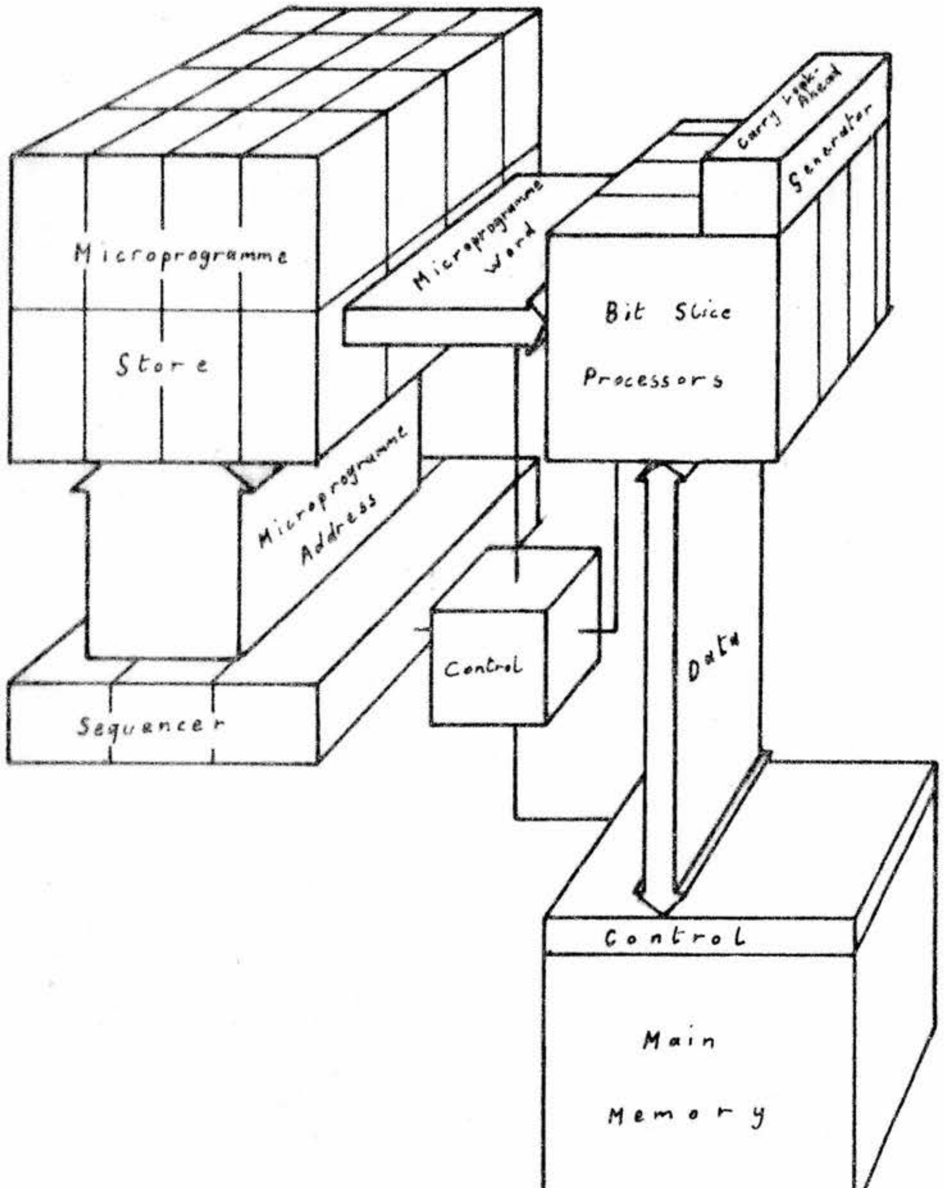

Microprogramme
Store
Microprogramme Word
Carry Look-Ahead
Generator
Bit Slice
Processors
Microprogramme Address
Sequencer
Control
Data
Control
Main
Memory

Figure 1

1.2 Simulation

It would be possible to design a system, burn in the read only memories and assemble the whole computer prior to proving any part of the hardware or software, however, even with the best designer, the probability of it working would be vanishingly small. A practical approach is to progress through various levels of simulation before committing the microprogramme to read only memory and assembling the complete computer. Not only should it be possible to modify the simulated system with ease, but also monitoring facilities should be available in order to debug and prove the system.

A first step in simulation is a purely "soft" approach. Computer programmes are written to simulate the logical functions of the various elements of the system, and then used to try microprogrammes and programmes. With a well structured programme it should be easy to prove the concept of a system, unlimited monitoring functions may be incorporated and modifications to the computer system are made by simple edits to a programme or its data.

There exist two main fallacies in hoping that this "soft" simulation totally mirrors a computer made of bit slice integrated circuits. The computer programme might perfectly represent the programmer's understanding of the printer's impression of the technical author's conception of what happens in an integrated circuit, but it is by no means unknown for small, yet significant, details to become corrupted along the line. Printers, for example, have the unfortunate habit of omitting inverting signs above variables. Thus the simulation programme could easily contain errors even if it is correct in the sense that it performs as the programmer intended. Moreover practical details, such as the finite, albeit measured in nanoseconds, delay generated by integrated circuits, are likely to be overlooked. A typical computer may have four seperate clock signals to ensure proper synchronisation, and to allow for propagation delays in components. To involve this level of detail in a "soft" simulation, it would make the programme too unwieldy and slow in running.

It is contended that the implementation of a further level of simulation before contemplating the construction of a complete system is highly advisable. This level should be closer to the hardware and incorporate the bit slice devices as far as possible to avoid any discrepancy between the devices and their description. Good monitoring systems and reasonable ease of modification should be retained.

One such technique involves the construction of a whole system, probably using standard modules for main memory and similar items, but substituting erasable read only memories (EPROMs) for programmable read only memories in the microcode memory. There are various types of EPROM, but in a common design the memory is programmed using voltages far higher than present under normal (reading) conditions, after which the power may be switched off and the integrated circuit behaves like a read only memory. These EPROMs may be erased by prolonged exposure to high intensity ultraviolet light or X-rays. The simulation of a programmable read only memory, however, is not quite perfect. The access time, the delay between a request for the value at a memory location and the value appearing on the output pins, is an order of magnitude slower with EPROMs than with PROMs. The clocks, instead of running with a period of about 100nS, will be slowed down to, say, 600nS. Monitoring and debugging facilities are not all that easily included without the addition of extensive supplementary circuitry. This would detract from the intrinsic desirability of a system where one merely substitutes erasable memories for permanently programmed ones and reduces the clock frequency.

In the totally "hard" approach a tremendous amount of work is involved in the construction of hardware which is neither as adaptable as it might be, nor does it simulate perfectly the microcomputer in view. If one is confident about the design of the computer, and wishes to use the machine in practical problems it is ideal; however for early investigations of different structures and microprogrammes it is not perfect.

Another technique, the one adopted by the author, involves dividing the simulation into two areas, one part performed by the actual integrated circuits under consideration, and the remainder by some host computer system. The host computer provides a wide, parallel I/O port into which the processor and sequencer integrated circuits may be plugged directly. The I/O port may be programmed to appear as any elements of the microsystem. Obvious candidates for simulation on the host computer are main and microprogramme memories. Assuming a conventional architecture, the I/O port would represent the microprogramme address bus, microprogramme data bus, main memory address bus, bidirectional data bus plus various control lines.

Any number of monitoring and debugging facilities could be included, as in the "soft" simulation. Timing is totally different from that of a real system - the response of the port is at least two orders of magnitude slower even than an EPROM - however provision is made for running the system under test for a burst of, say, 100nS and then leaving it dormant until the port is ready for the next cycle.

While the port is designed principally as a device into which a more or less complete central processor is plugged, the hardware is suitable for many other uses. The most relevant is the examination of single integrated circuits which are used in the microcomputer assembly. A sequence of signals may be applied to the inputs and the outputs tabulated. Once a design is settled upon, it may be desired to build a complete system. At this stage devices such as memory and I/O interfaces can be tested through the port.

1.3 Interface Requirements

Table 1, due to Roger Haynes, defines the number and function of the lines required between the host computer system and the bit slice devices. Present length refers to a system of his design. This author feels that additional control lines, such as handshaking, or memory ready, protocols and interrupt inputs, might be required.

Field	Direction	Present Length	Estimated Maximum
Microcode Address	->	12 bits	12 bits
Microcode Word	<-	28 bits	<40(probably <32)
Main Memory Address	->	16 bits	16 bits
Main Memory Word	<->	16 bits	?
Control: clock	<-	-	1 bit
read/write	->	part of microcode word	

Totals: Microprocessor -> Host System		28 bits	28 bits
Microprocessor <- Host System		29 bits	<41 bits
bidirectional	<->	16 bits	?

Table 1: Bit Slice Microprocessor / Host System Requirements

It was seen that something of the order of 100 lines were required. The main computer's word width, and hence the width of standard parallel interfaces, is considerably less than this, so some form of multiplexing and latching was required. The bit slice microprocessors are designed to operate in a T.T.L. (Transistor Transistor Logic) environment, so it seemed that the ideal integrated circuits to form the elements of the port should be from the industry standard 74 series T.T.L. range.

The output circuit of a normal TTL gate is, essentially, two transistors, one connected between the 0 volt supply and the output terminal, and the other from the output terminal to the positive supply (figure 2). Under normal operating conditions either one or the other transistor is free to conduct electricity while the other is off, thus the output is either clamped well down to 0V or actively pulled up to the positive supply. This is known as a totempole output. In one variation of a TTL gate, the tri-state, there is a further input beyond the logic inputs, termed the output enable. In the active state a tri-state gate acts like a totempole output gate, however if the output is disabled both transistors in the output stage are turned off and the output is free to float to any reasonable voltage. This high impedance state does not depend on the input logic levels.

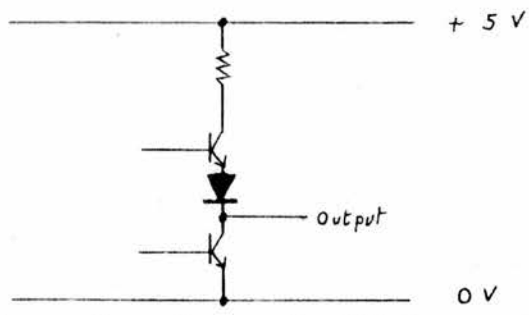
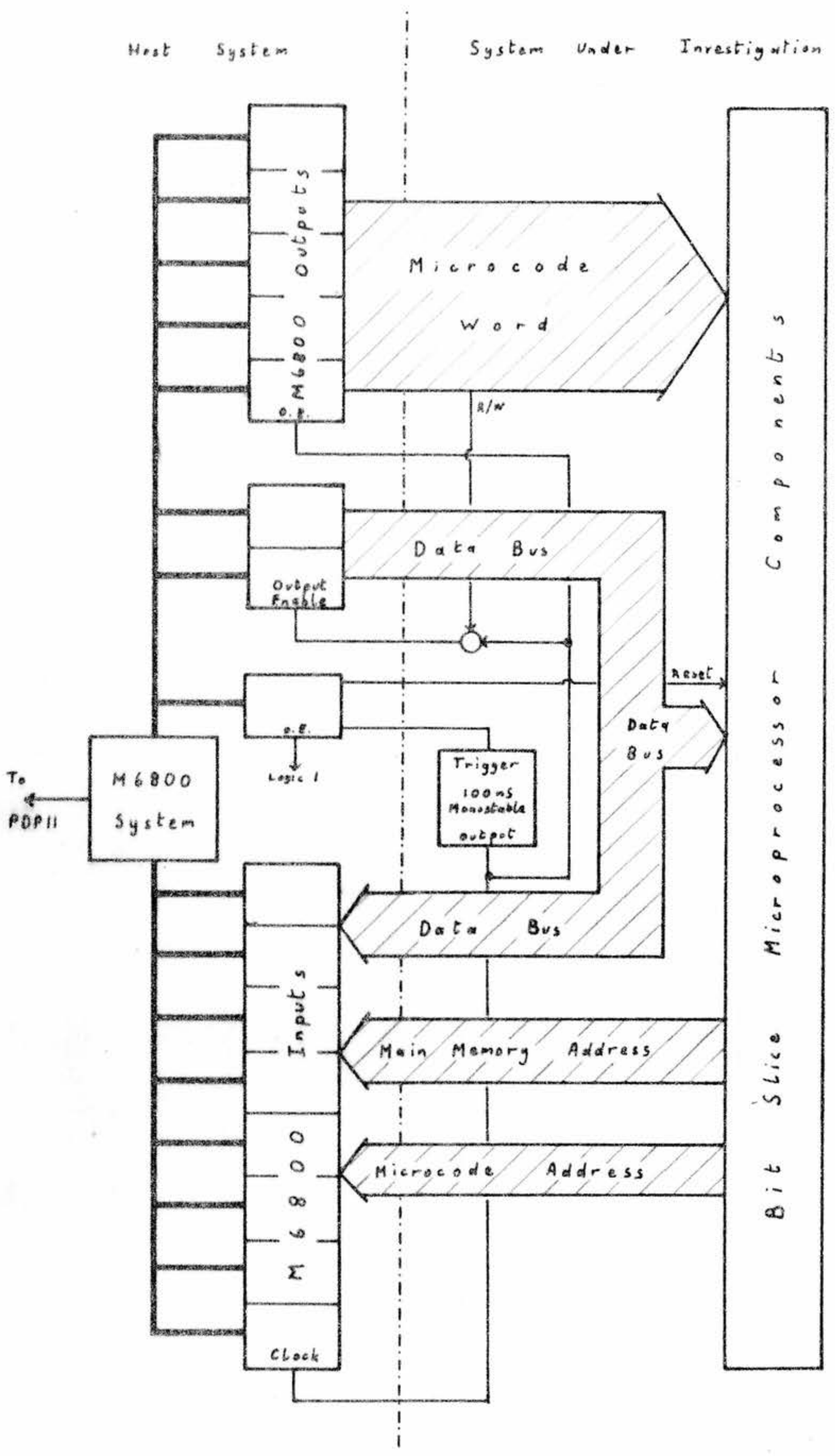


Figure 2: Totempole TTL Output Stage.

Above it was stated that it was desired to have the bit slice devices under evaluation enabled for a brief period and then lie quiescent until the host has serviced them again, in order to give some impression of true operational timing. Outputs from the port are tri-state and only enabled briefly when signalled that valid new data are present. The effect of input signals to the port is frozen in latches (clocked) a short time after the outputs are enabled.

It was decided that a versatile and practical system would be to have 64 input and 64 output lines from the port, each bank of 64 being divided into 8 groups of 8 lines. Each group, or byte, could be enabled or clocked separately. A typical system might have one output byte from the port enabled permanently to transmit status information (clock, reset etc.). The clock would have an active period greater than that required, so it would trigger a monostable (a circuit that produces a standard length of output pulse for any length of input pulse) to form the actual clock presented to the bit slice microprocessor. The same signal, or its derivative, would clock/enable the other bytes. A bidirectional data bus could be formed by connecting together, say, the sixteen lines from two input and two output bytes. The microprogrammable devices' read/write line would be used to determine whether the output bytes should be enabled in any particular cycle. This configuration is illustrated in figure 3.

One view of the port has now been considered - the main host computer's is yet to be dealt with. There are two means of on-line communication with a computer - through standard interface devices supplied by the manufacturer or through a purpose built interface that taps the computer's buses directly. This latter option was ruled out on two grounds, viz interference with other users through any un-intentional influence upon the operation of the existing hardware or



Host System
System Under Investigation
M6800 Outputs
O.E.
Microcode Word
R/W
Output Enable
Data Bus
Reset
O.E.
Logic 1
Trigger
100nS Monostable output
Data Bus
To PDP11
M6800 System
Data Bus
Inputs
M6800
Clock
Main Memory Address
Microcode Address
Bit Slice Microprocessor Components

Figure 3

modifications to the operating system software, and the time, expense and difficulty involved. Standard interface devices fall broadly into two categories - serial and parallel. The latter would have many advantages such as transfer speed and convenience, however modifications would have to be made to the operating system to allow for the new interface, which would itself have to be purchased. Serial interfaces were already present on the host computer, and they had considerable advantages. They are driven by tested software and easily accessible from high level languages, they have electrical connections that are very tolerant and they can be made to operate at reasonably fast rates, but do not monopolise the host computer. If the I/O port has no intelligence, one could hope for something betwen 10 and 50 microcode cycles per second. This may sound unimpressive, but it is comparable with the rate of "soft" simulations. Further, as already noted, the designed operating timing is simulated.

In order to attach the latches of the port to the standard serial interface a fairly large amount of random logic would be required, so it was decided rather to build a system based on a microprocessor, allowing some intelligence to be incorporated in the serial to parallel conversion. Provision is made for some buffering which allows a cache memory for part of the microprogramme store. Checksum facilities are available to aid in the detection of transmission errors. It is assumed that in the majority of microcode cycles that the only reference will be to another, nearby, microprogramme word, so greater independence, and, hopefully, speed will result if a block of microprogramme is stored in the interface. In general a more versatile system has been created which, due to its "stand alone" functions, should reduce the load on the communications channel.

The decision to use a microprocessor based interface introduced a new phase into the project - software development for the interface's microprocessor. This involved both the introduction of a cross-assembler into the main computer and its use.

1.4 Summary

The design objectives of the system were to provide an interface between a host computer and the elements of a bit slice microprocessor with a view to analysing the operation of the latter. The interface was required to be such that there was minimal interference with the host system and the appearance of the interface to the microprocessor under evaluation resembled as closely as possible the remaining parts of a self-contained microcomputer. Realisation is through industry standard integrated circuits for connection to the bit slice devices and communication with the host is by serial signals through an existing multiplexer also used for computer terminals. Routing and control within the interface is performed by a microprocessor.

CHAPTER 2

Existing Material

Of the two main items of hardware in the microprogramme development system there was no choice in one. The only computer available to act as host was a PDP11/40 running under the Unix operating system. A range of suitable microprocessors on which to base the interface was on the market, however the author had experience of the Motorola M6800, and the offer of the loan of its development kit was made, so it was the obvious choice.

2.1 PDP 11 Hardware

For the purposes of this exercise the only important part of the PDP11 hardware is its asynchronous communications interface. The model available was a DJ11-AA 16 line asynchronous serial line multiplexer with V24 electrical interface(1). Its principal use was for communication with computer users through terminals.

Lines from a DJ11 are divided into four groups of four lines. Each group may be preset to any popular baud rate (75 to 9600 baud) and format of data bits, parity and stop bits. The V24 electrical interface is the standard C.C.I.T.T. voltage switching levels to represent marks and spaces (equivalent to the American RS232).

2.2 PDP 11 Software

The software used on the PDP11 is written in a language called C(2) which shows much influence from B.C.P.L. but has unusual constructions, especially for iteration. Emphasis is placed on pointer manipulation. C is designed with systems programming in mind, hence it has all the bit manipulating facilities etc. that one would expect to find in an assembly language, however it retains the ease of use of a high level language once its notational peculiarities are mastered.

The interface with the operating system, UNIX(3), is exceptionally good; there is no difficulty, for example, in opening or creating files from a C programme or calling systems routines to alter the characteristics of serial I/O lines.

2.3 M6800

The Motorola M6800 system(4) is a series of supposedly compatible integrated circuits that are used to form a microcomputer. The principal device is the MC6800 microprocessor which is an 8 bit CPU with an address space of 64K. Read only memories, random access (read/write) memories, parallel and serial interfaces are also available. All input/output is performed by reference to memory locations.

Motorola produces a development kit for the M6800, the MEK6800D1(5), which includes an MC6800, some RAM, peripheral interface circuits and a ROM, the MCM6830L7(6), preprogrammed with various utility programmes. The package of utilities, termed Mikbug by Motorola, is intended to communicate with an operator through a teletype allowing him to load programmes into RAM, list the contents of registers, examine and modify memory locations etc. A circuit diagram for the kit as modified by the author is shown in appendix 1(ii). The only significant modifications are the addition of a line to the edge connector that is used to enable the ROM, and changes to the serial communications circuitry to allow the use of the much higher baud rates preferred for communications between machines. The system clock has been slowed down to give a period of 600nS for ϕ_2, thus relaxing the speed constraints on the memory.

Communication with the development kit is through an asynchronous line. For off-line work and the development of on-line communication with the PDP 11 a scrap teletype was purchased. The printer required its code converted to ASCII and the electronic keyboard was completely rebuilt. While the printer is restricted to 110 baud operation, the keyboard may either use an internal clock for 110 baud or an external clock for virtually any other rate.

CHAPTER 3

New Hardware

The new hardware in the system may be divided into two parts - permanent features of the system and development aids. The former category includes the parallel interface to the microprogrammable devices, a module of additional memory and a mother board into which everything including the M6800 kit is connected. An exchange unit (for routing serial signals) and a V24 interface unit (for conversion of the microprocessor signal levels to those of the PDP 11) were built for test purposes, but simplified versions could be used in the final unit.

Circuits for all parts of the system are illustrated in appendix 1, with a brief explanation of the notation in the first part. Details of the connections are available in appendix 2. A "worst case" timing diagram, drawn specifically for the R.A.M. circuit is reproduced in appendix 3.

3.1 Mother Board

All circuits are connected to the mother board, which comprises of a number of edge connectors for added circuit boards plus some buffer circuits to amplify signals. Apart from the microprocessor board, which is a slightly modified MEK6800D1, it was decided to adopt a bus structure with all edge connectors identical, allowing full interchangeability of new circuit boards.

For simplicity in the design of custom circuits the bidirectional data bus from the MC6800 is split into two - an output bus from the microprocessor and an input bus to the microprocessor. The data and address outputs are maintained on the mother board buses for longer than is guaranteed by the microprocessor, so circuits have longer to respond to them.

Appendix 1(iii) shows the circuit of the mother board. It is seen that the circuit comprises of three blocks plus some controlling gates. The bottom two blocks are used to split the microprocessor's bidirectional data bus into two unidirectional buses while the upper one is buffering the address bus.

In order to reduce the speed constraints on peripheral interfaces and memory it was decided to store the last Valid Memory Address from the microprocessor on the address bus of the mother board until such time as a new Valid Memory Addesss was issued. This allows the maximum possible time for bus devices to respond to their addresses. The latches (or registers) used have the advantage that in addition to the normal output (Q), a complementary output ($\bar{Q}$) is available. The eleven most significant of these form part of the mother board address bus to simplify decoding. By performing an AND function of various bus lines blocks of 32 bytes may be uniquely identified. It was envisaged that each board plugged into the mother board would generate a select signal (to identify when it was being addressed) by gating these lines with V.M.A.. Considering the timing diagram, it is seen that the remainder of the address lines are set up prior to the start of the select signal (marked $\overline{CE}$) and held to the end of it, a requirement frequently found in integrated circuit specifications.

A reduction in the number of pins of the MC6800 integrated circuit is achieved by making the data bus bidirectional. Width of the mother board bus is not so critical, while timing considerations can be of importance, so it was decided to split the data bus into two, one output from the microprocessor and another eight bit bus input to the microprocessor. The output bus is latched (stored) so valid data are held beyond any select signals used by the devices on the mother board bus.

Output from the microprocessor is available when the R/$\overline{W}$ line is low. The microprocessor data bus is enabled by $\emptyset_2$ on the microprocessor board, so valid output is transmitted by the data bus during $W \wedge \emptyset_2 \wedge$ VMA, which signal is used to enable a latch driving the output data bus.

Input to the microprocessor from the mother board devices comes through a tri-state buffer. This must be removed from the high impedance state to transmit data from the mother board input bus whenever the microprocessor is reading (R/$\overline{W}$ is high) and a device on the mother board is selected. One line of the mother board bus ($\overline{\text{selected}}$) is held low by any circuit that is addressed, failing which it floats high. Together these signals are used to enable the buffer.

3.2 2K Memory Board

At the time of construction the most economical manner of buying Random Access Memory was in integrated circuits of 1K by 1 bit. It seemed reasonable, therefore, to build two independent blocks of memory, each 1K by 8 bits, that could be switched to start at any 1K boundary in address space. Switches are also incorporated to inhibit writing to each block, providing a Read Only Memory simulator. A flexible system is thus created where, say, a programme may be loaded into one block, after which it is set to write protect, while the other block is used for data storage.

No restriction exists on when one may change the address allocation of a block of memory, save when addressing the block in question. It is quite possible to write to a block of memory at one address, write protect it, and then change its base address. Typically this would be done if it were desired to load a replacement for the Mikbug using the Mikbug itself. Once loaded the address of the memory would be set to $FC00_{\text{hexadecimal}}$ and the Mikbug R.O.M. disabled; any interrupt would take its vector address from the R.A.M. and the processor would proceed accordingly.

The memory integrated circuits chosen were Signetics 2102-1. It is important to stress the manufacturer as specifications for nominally equivalent memory circuits vary considerably from one manufacturer to another. This integrated circuit is a static (i.e. data are stored in bistable elements and valid as long as power is applied, without any need to refresh the contents) random access memory with a cycle time for read or write of 500nS. Inputs to each circuit are:

1 10 address lines.

2 A read/$\overline{\text{write}}$ line - logic level 0 for write.

3 Datum in.

4 $\overline{\text{Chip}}$ $\overline{\text{Enable}}$ ($\overline{\text{CE}}$) - a logic 0 applied to this input indicates that the integrated cicuit has been selected to be active.

In order to write to the integrated circuit these lines must be set up in a specific order with closely defined minimum times between them. The sequence of operations is:

1 Set up the address lines.

2 Apply a $\overline{\text{CE}}$ signal.

3 Apply a write signal.

4 Set up the required input datum.

5 Remove the write signal.

6 Remove the $\overline{\text{CE}}$ signal and, if desired, the address.

7 Finally the input datum may change.

The read cycle follows a similar pattern.

The circuit of the R.A.M. (appendix 1(iv)) is extremely simple with the exception of the R/$\overline{\text{W}}$ circuitry. An $\overline{\text{enable}}$ signal is generated by gating switched selections of the top six bits of the address bus and their complements with V.M.A. plus another signal that is normally a logic level one. Here is found the only asymmetry in the circuit: in one half of the 2K memory the signal is permanently wired to the positive supply, while the other half derives its eighth input from the $\overline{\text{enable}}$ output of the first 1K. Thus both banks of memory are precluded from responding to the same address, even if the switches are set accordingly. This signal is used for the $\overline{\text{CE}}$ of the R.A.M. circuits and, through a diode, as a signal to enable the input buffers on the mother board on microprocessor read cycles.

Appendix 3 shows a timing diagram for the R.A.M. circuit. This illustrates the worst case situation where all the components conspire to take as long or short as possible to function, whichever is less convenient. The original design of the R.A.M. circuit was based on a timing diagram found in the Motorola M6800 Applications Manual, however this was modified since their timings were not truely worst case - they were slowest case rather. Further the specification for the MC6800 has changed since the advance information used for the Applications Manual. All the signals are related to the two anti-phase system clocks $\emptyset_1$ and $\emptyset_2$. The lines appear more or less as they might be seen on an oscilloscope with low (logic 0), high (logic 1) or floating values. Sloping lines indicate that the transition can occur at any point along their length.

On comparing the signals available on the R.A.M. board with the specifications of the R.A.M. it is seen the $R/\overline{W}$ line requires considerable modification from the mother board signal. The onset of the write signal is delayed by a monostable of period 80nS and truncated such that it returns to its normal high state by the end of $\emptyset_2$. A switch is provided that removes the possibility of writing to memory by holding the $R/\overline{W}$ line high.

It should be noted that although the outputs of the 2012 R.A.M. integrated circuits are enabled whenever the chips are selected, signals are not transferred from the mother board bus to the microprocessor bus except during microprocessor read cycles.

3.3 Parallel Interface

A block diagram for the parallel I/O interface is shown in appendix 1(v). Selection of the block of 16 addresses is performed in a manner similar to the 1K memory allowing the interface to be located on any boundary of 16. Output from the select gate is used to select an integrated circuit which operates as a one of sixteen decoder on the four least significant bits of the address bus. The decoder provides sixteen lines responding to unique contiguous addresses switchable within the address space.

Latches are in turn connected to the sixteen lines from the decoder. Input (to the microprocessor) latches are clocked by the microprogrammable devices and their tri-state outputs are enabled onto the mother board bus when selected by the M6800. Output latches have data clocked from the mother board bus when selected - their outputs are, however, enabled by the external devices.

The latches used are termed transparent latches. In other words the outputs (Q) follow their data inputs (D) whenever the enable is high, retaining the value at the negative transition of enable signal. In retrospect it has been realised that it might have been better to use edge triggered flip-flops on the output of the M6800. On positive transitions of the clock the Q output is set to the logic state of the D input. Under most circumstances these will have the same effect in the interface, however, when using transparent latches, spurious transitions could occur during M6800 write to interface cycles. The microprocessor data bus is not set up until T_{ASD} (about 150nS) after the latch is enabled.

Only two 8 bit inputs and two 8 bit outputs have been constructed.

3.4 Exchange

All serial signals are routed through the exchange. The exchange consists of six columns of four, three position switches. At the foot of each column, except one which is linked to the printer, is a socket for connection of serial communication devices. In effect the exchange is four horizontal lines which may, at each node, be switched to either the transmit or receive line of the connector below, or may be left open circuit. Any number of transmitters and receivers may be connected to any horizontal line, but characters will, of course, become corrupted if more than one line attempts to transmit simultaneously (however no damage will be done).

Appendix 1(vi) contains the circuit diagram for the exchange. Signals are buffered on entry to and exit from the exchange. Stop signals are represented by a logic level one, hence in the idle state the cathodes of the diodes are floating to a high potential, but when a start level is applied current is sunk through the diodes from any NAND gate switched into the circuit. Thus all the inputs are no longer floating to a logic level one and the output changes accordingly.

3.5 V24 Interface

Communication with the PDP11 is through the TTL interface box. This unit includes a converter for different signal levels and protection devices for the PDP11 in case of failure in any component on the microprocessor side.

The circuit for the interface is extremely simple (appendix 1(vii)). It consists merely of two level convertion integrated circuits, one capacitor (to limit the slew rate on the V24 level output) and protection components based on a Post Office design(7).

3.6 Constructional Method

There are four main units in the system, viz the V24 interface, exchange, mother board and microprogrammable microprocessor connector. The V24 interface and exchange are built in separate boxes, however the mother board is merely bolted onto a sheet of aluminium and left open for easy access. The electronics for the mother board is built on a standard sheet of Veroboard, the parallel conductors forming the buses and accepting the edge connectors directly. From the parallel interface card come ribbon cables terminated with an edge connector mounted on a piece of aluminium. Into this edge connector is plugged the microprogrammable microprocessor.

Connections between components on the memory and interface boards are performed in a novel manner - the Verowire system. This is a rapid technique that is a cross between wire wrap and soldering. Components are mounted on a board which is printed with a regular array of copper pads and their leads are bent at an angle to the board. Wire, enameled with self fluxing varnish, is then wrapped round one lead and guided along special wiring combs to its destination where, again it is wrapped. Finally the joints are soldered, the varnish boiling off at soldering iron temperatures. This is the fastest wiring method known to the author, and he found no difficulty in its use. The wires look somewhat vulnerable to damage, and their close proximity could lead to cross-talk, however no evidence was found for either.

CHAPTER 4

Software Concepts

The aim of the microprogramme development system is to provide a tool with at least two facilities. First the operation of the elements of bit slice microprocessor may be investigated. Thereafter, once a system has been designed and microprogrammes written, they are tested in conjunction with each other with the aid of the unit. The hardware remains the same, however its function is tailored to suit the application by software.

Completely different situations exist in the two cases. In the former stage a completely passive specimen is under the microscope and the response to various isolated stimuli is examined, whereas in the latter case the observer is allowing a live animal to run round its world by itself. He is no longer directly controlling the stimuli - he merely initialises the microprocessor's world and tells it when he is ready for it to take a step. Roles have been swapped - formerly the units attached to the port were sub-peripheral in status, but in the latter use of the system they have taken over processor status and the M6800 and PDP11 function more as peripherals.

The two roles require contrasting types of programme for the M6800 and the PDP11. In the initial bit slice device evaluation stage the M6800 system has little to do except convert serial signals from the PDP11 into parallel lines to the port and vice versa. During microprogramme development, however, with bit slice devices forming a complete processor, the M6800 system becomes a storage manager and the PDP11 a bulk store and I/O processor for operator communication.

Software for the microprogramme development system may be divided into three sections, viz. that associated with the microprogrammable microprocessor itself, programming the interface and programming the PDP11. Of these, the first is outwith the scope of this project, the second has been facilitated by the procurement of an assembler, and an extensive range of communication routines has been developed to form a kernel for all programmes on the PDP11.

4.1 M6800 Assembler

It was intended to write a cross assembler to run on the PDP11 for M6800 code, however shortly after work was commenced, the author was given a copy of the source for an assembler written in C. This was compiled, its method of use established, one bug corrected and it is now available on the PDP11. A reference manual(8) and a system description(9) are available.

4.2 Communication Routines

Communications between the M6800 and the PDP11 are of fundamental importance to the project. Use is made of the MIKBUG(6) routines on the M6800, and these are complemented by mirror routines written in C for the PDP11. For example, the C routine read_registers communicates with the MIKBUG routine PRINT to determine the value of the M6800 registers. These values may be altered in the PDP11 and then transferred to the M6800 using restore_registers which, in turn, communicates with LOAD in MIKBUG. A detailed description of each C routine is given elsewhere(10).

4.3 Other Modules

Two other modules exist to simplify the writing of programmes on the PDP11 for use with the M6800. One is used to declare various standard variables used in the communication routines, such as the PDP11's representation of the MC6800 registers, and the other sets up parity and other options (stty parameters in Unix terminology) for the line to the M6800.

4.4 Summary

It is seen that in the early stages of work the human operator investigates the bit slice devices through the PDP11 and M6800, while in the later stages the bit slice devices happily work by themselves, asking for information from the PDP11 store via the M6800; the human operator passively watching what happens. The key to operation of the units is software, which is based on the PDP11 where cross assembler and communication programmes are available.

CHAPTER 5

System Testing

Before considering using a tool for development purposes the tool itself must be thoroughly tested. There is no point in utilising a debugging aid if half of the trouble is caused by the aid itself. Each part of the system has been subjected to various tests; these are described in a certain amount of detail as they illustrate the use of the system, as well as their nominal function.

5.1 Communications Hardware

The first stage in testing the system, once operation of the M6800 development kit in conjunction with the teletype and exchange was established, was to check the communications hardware (serial input/output of the M6800, exchange, V24 interface and cabling) at various baud rates. Trivial programmes were written for the PDP11 to establish that signals may be sent from the M6800 to a terminal of the PDP11 and vice versa. Data rates tried were 110 baud, 300 baud, 4.8k baud and 9.6k baud, all with success.

5.2 Communications Software

Mirror routines for the I/O routines of the MIKBUG were developed and tested by monitoring both sides of the communication on the teletype. A typical example is the operation to load the copy of the MC6800 registers stored in the PDP11 into the MC6800 (restore registers). Trial values would be set up on the PDP11 and the mirror routine would load them into the MC6800, all signals being monitored by the teletype. As a further check, these values could be read back either via the PDP11 or manually by switching the keyboard to communicate directly with the M6800.

The reliability of the communication software and hardware was proved through extensive use during memory and interface testing.

5.3 Memory

It may be argued that one of the most critical aspects of a computing system is the reliability of its memory. Accordingly, the 2K RAM board was repeatedly tested via programmes running on the MC6800 with operator communication through the PDP11. Some proof of the validity of the test was given by the fact that three types of error were detected. First, due to a misprint in the MEK6800D1 guide, there was a conflict between the addresses to which the 1K memory blocks were switched and other devices on the bus and this, understandably, caused problems. Errors were intentionally induced by removing a single memory integrated circuit, and by switching a bank to read only, both of which were detected.

After manual testing of random locations of the new memory the design of an automatic test was considered. Various types of test may be used, varying from checking that 00 and $FF_{hexadecimal}$ may be written into each location to extremely long tests where, for each possible value in each location, any modification of any other location has no effect on the stored data. A compromise was settled upon where it was repeatedly tested that any value may be stored at any location. Each cycle of this short test lasted about 15 seconds.

A complete listing of the memory testing programme is given in appendix 4, not only for its intrinsic interest, but also as it is illustrative of the general approach adopted. The listing commences with some declarations and initialisations, largely standardised. 6800 code from the file given as a parameter in the programme call is first loaded down to the M6800 system, then the registers are initialised, read back and displayed. Once the 6800 is ready, it is instructed to Go and the PDP11 waits for an opportunity to read the MC6800 registers - either caused by an error condition or by the test being completed. In the former case the error is noted and the user has the option at every third error of giving up, carrying on or moving to the next byte. Completion is indicated by the contents of the index register being outside the range of the memory being tested, in which case the total number of errors detected is indicated. Finally all signals are passed through the PDP11 to the user for manual intervention, if desired.

As a simple check on the practical use of the 1K RAM, it was switched to the address of the scratch RAM used by the MIKBUG ($A000_{16}$) and the RAM supplied in the kit removed. The system functioned correctly.

5.4 Parallel Interface Test

The simplest way to test a parallel I/O interface is to put signals out on one channel, loop them back to another, and read the result. A board was constructed to plug into the microprogrammable microprocessor connector with random one to one connections between the input and output lines and was used to test the interface.

It was during the testing of the parallel interface that trouble was first experienced with the serial communications. A large table, generated on the M6800 was to be printed through the PDP11, however many of the characters were lost on the way. Investigation showed that if a burst of more than about 64 characters was transmitted by the M6800 at high baud rate, most of the characters were lost, but the last 64 remained intact.

The DJ11 multiplexer has an internal first in, first out buffer (FIFO) with a capacity of 64 characters. It was considered that this could be overflowing due to slow response from the operating system software. Attempts to monitor the operation of the multiplexer driver by printing messages on the control console proved futile since all other processes are halted during system error message printing.

A trivial driver for the multiplexer was written that transmitted all characters received on the microprocessor's line to a V.D.U. operating at the same baud rate as the microprocessor. This proved that there was no difficulty in the PDP11 hardware coping with transferring data from one line to the other at high data rates or with the M6800 hardware. The PDP11 ran quite normally (with the exception of terminals connected to the multiplexer) while the microprocessor was continuously transmitting at 4.8k baud.

If the hardware was not causing the problems, they must be connected with the software. Further investigation was carried out into the operating system's serial communications software, and this revealed that if the internal software buffers are filled to overflowing, their contents is thrown away. As an experiment, the appropriate buffers were quadrupled in size to check that this was the difficulty. In practice this modification was sufficient for the present applications.

5.5 Total System Test

The elements of the system had been tested separately, but they had to be tested together in case of any unpredicted interaction. Both new boards were plugged into the mother board and the tests repeated. Except for some noise, cured by the introduction of a small capacitor, no trouble was experienced.

5.6 Summary

One cannot prove conclusively that a piece of hardware will work in every situation - that requires total acceptance that the devices satisfy their specification and ignoring all uncalculated factors such as coupling between adjacent circuits - but one can, by showing that it cannot be faulted, satisfy oneself that it may be assumed to work. While it has not been possible to use the system to drive a microprogrammable microprocessor system, the elements of the development system have all been tried either explicitly (e.g. the RAM board) or implicitly (e.g. the mother board).

CHAPTER 6

Future Work

In order to bring the development system into full and convenient operation further work must be completed. This may be divided into two areas, viz original work and tidying up. The latter involves extension of the parallel interface to its full complement of 2 by 64 bits, and replacing the separate exchange, power supplies etc. with a single "black box" which provides all these services.

6.1 Memory Allocation

Table 1 describes a system capable of addressing 151552 x 8 bits of memory - about 1% of that can be buffered in the RAM of the M6800 system. Optimum advantage of the buffering will depend on the design of the bit slice microprocessor. A balance must be found between the access speed advantage of finding the required datum in the M6800 memory and the overhead involved in transferring large quantities of data from the PDP11 to the M6800 which may be overwritten in the next cycle, never having been utilised.

Many accesses are made to the microcode memory for each access to the main memory, thus it seems worthwhile to use the buffer space largely for microcode. The allocation will depend on the bit slice devices used and the way in which the microprogramme is written.

AM2900 series devices are microprogrammed in a fairly conventional manner - logically consecutive micro-instructions tend to be arranged consecutively in memory. A solid block of microcode memory, or several solid blocks including much used code and a block that can be overwritten, is therefore the choice for Advanced Micro devices. 256 microcode words (1K bytes - as much space as could be spared) would be too for large a single block (taking a second to load from the PDP11), while the overhead involved in loading a few words would be excessive.

Another popular bit slice system, the Intel 3000 series, has a completely different microcode sequencer. There is no automatic incrementer for the microcode address, so there is no reason for consecutive micro-instructions to be arranged consecutively in memory. The microcode memory may be considered to be laid out in rows and columns. Row 0 can be accessed from any instruction, columns 3 and 11 from successful conditional jumps, columns 2 and 10 from unsuccessful jumps and certain other jumps require a destination in columns 12 to 15. In addition it is possible to jump to another address in the same row or column. Thus it would appear that certain areas of microcode memory will be used far more than others. A reasonable use of the buffer might be to store permanently row 0 and columns 2,3,10,11 and, space permitting, columns 12 to 15. If the required microcode was not in the buffer, the appropriate column would be loaded from the PDP11 into a scratch area of RAM.

The programme space of the main memory is likely to be accessed sequentially in units of a few bytes at a time. The minimum area to be buffered should be capable of holding the longest instruction possible, while it would be an advantage to buffer several instructions.

Data storage will depend very much on the architecture adopted for the bit slice microcomputer. If, say, the first n locations of memory are regarded as registers with fast access times or that may be addressed with short instructions, they should be held permanently in the M6800 system. Other data would have to be held basically in the PDP11 with possible buffering in the M6800. How valuable this would be is open to debate - data storage is read/write memory, unlike the programme and microprogramme stores, so all data must be copied back into the PDP11 again after being operated upon by the bit slice microprocessor. With stack based designs, many of the memory accesses will be to sequential locations, so it may be worth keeping the top few locations of the stack in the M6800.

Programme storage for the M6800 will consume a considerable quantity of space, more space being required to handle the complex buffering for devices like the Intel sequencer. In addition to the arrays of data buffered for the bit slice microprocessor and their labels to indicate what they represent, various accounting variables would be stored. These would measure microcode and main memory accesses to enable estimates of performance in real systems to be calculated.

6.2 Coroutines and Pipes

At present the process running on the PDP11 must know whence the next input is coming - the M6800 or the control terminal. It would be more satisfactory if normal communication with the M6800 could be non-destructively interrupted by the operator at the control terminal to enquire of the current status of the simulated bit slice microcomputer.

Within the Unix operating system there are specified two facilities termed coroutines and pipes. The former enables logically parallel operations to be executed logically simultaneously. Pipes allow inter process communication, and may be joined with "tees". If these facilities were made available and used on the PDP11, parallel processes could be established, one communicating with the M6800, and the other with the operator.

6.3 Summary

Half of the work requiring to be completed on the project is connected with the elimination of all the switches and bits and pieces of electronics. The hardware development phase is completed and the controls on the development system box can be reduced to an on/off switch and a reset button. The other half of the work involves implementing a development system for use with a particular design of bit slice microcomputer. Expansion of the parallel interface is required to the appropriate number of bits, and a considerable quantity of software must be written both for the M6800 and the PDP11, making use of the existing communications routines.

CHAPTER 7

Conclusions

The original aim of the project was to provide and demonstrate a programmable interface between a microprogrammable microprocessor and a PDP11, along with all the required software. To this end all the fundamental building blocks have been proven, however since no microprogrammable device was available only a sample 16 bits of the interface was constructed and the device dependent software is not written.

It has been seen how the existing M6800 kit has been expanded to accept the additional memory and interface. Minor changes have been made to facilitate the use of higher baud rates. The system has been constructed without any modifications or additions to UNIX or the PDP11, save possible enlargement of the UNIX software buffers and alteration of the strap settings on the DJ11 multiplexer.

New hardware has been constructed, both as permanent features of the system and as development aids. The memory size of 2K bytes should be sufficient, with the use of the PDP11 as a backing store, for the present purposes, however further modules could be constructed to the same circuit and switched to other free areas in store if required. Provision for 64 input and 64 output bits in the parallel interface has been made, although only 2 x 8 bits have been assembled in each direction. The system is now at a stage where the exchange and teletype may be dispensed with and the system converted into a self-contained peripheral of the PDP11.

Software has been written with the aim of maintaining a dialogue between a programme running on the PDP11 and one on the M6800 through the Mikbug routines and parallel procedures written in C. At one stage it was hoped to programme the PDP11 to generate the M6800 programmes automatically, however time did not permit this.

The ultimate test of the system would be to use it to drive a microprogrammable microprocessor, however the complementary project failed to provide a suitable device. The testing routines have proved that no fault can be found with the elements of the system, whereas a positive demonstration of the system would prove more satisfying.

It is difficult to see how the system could be improved within the original specification. Increasing the memory size in the M6800 could improve real time operation of the interface, however this is not a prime consideration. The purpose of the interface is to provide a development system capable of demonstrating the functional capabilities of a microprogrammable microprocessor configuration with a minimum of dedicated hardware requiring to be constructed and the ability to incorporate debugging facilities. In order to measure the rate of operation of the microprogrammable device a count of its cycles would be kept, possibly with a weighting attached to simulate the operation of slow memories, and this would be divided by the clock rate in a practical system.

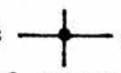
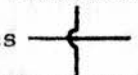
The intended use of the system is to simplify the simulation of computers based on bit slice microprocessors. No simulation can perfectly represent the object in view, however it can prove most valuable in its ease of use, adaptability, proven facilities and economy. There is no reason, however, why this system should not be put to other uses as intelligent parallel/serial converter interfaced to the PDP11. One such example would be the development of an interface for a high quality printer requiring parallel signals. The interface could be programmed to translate signals intended for a teletype into suitable codes.

APPENDIX 1(i)

Notation

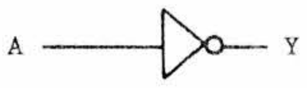
The notation used in these appendices is reasonably standard, however below are listed the symbols and functions for the various simple components. Special components are represented by boxes and marked with their type numbers.

Connections

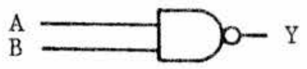
All single wires are shown as thin lines, with joins indicated by dots , and crossed wires without connection by bridges . Buses of several lines are illustrated by a pair of parallel lines with arrows to illustrate the sense of the signals (or if they are bidirectional) except on the processor board where, for the sake of space, the buses are represented by a single broad line.

Gates

Inverter, $Y = \overline{A}$



NAND, $Y = \overline{A \wedge B}$



Latches

All latches used in the project are transparent with positive enables. Their function may be represented by the table:

Enable (G)	D input	Q output
0	X	Q_o
1	0	0
1	1	1

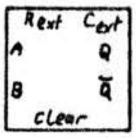
where X means that the signal is irrelevant and Q_o is the value of the Q output prior to the 1 to 0 transition of the enable input.

The latches used in the parallel interface (type 74S373) have an output enable control that disables the tri-state outputs when a high level (1) signal is applied.

Monostables

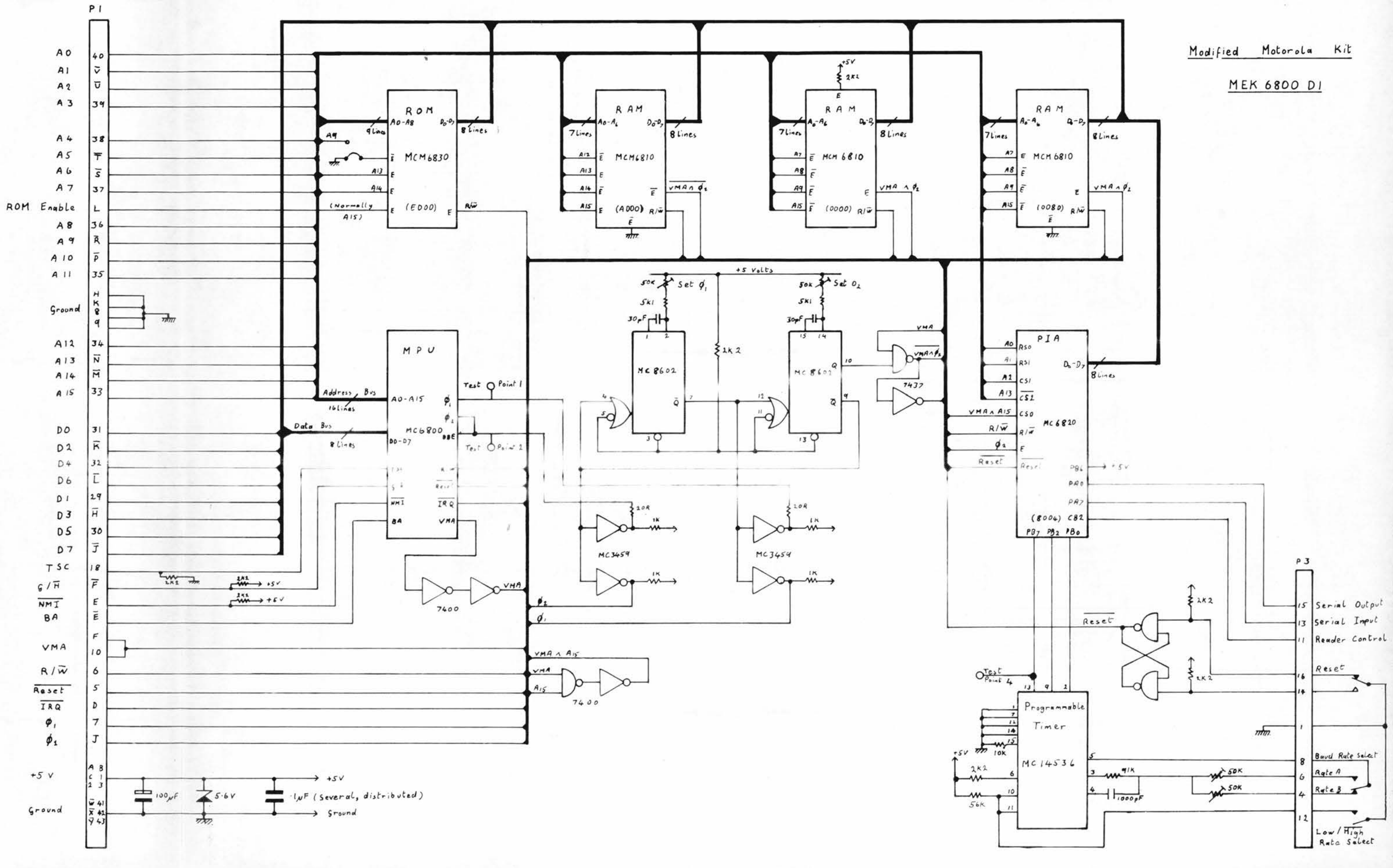
The period of the monostables is determined by an external resistor and an optional capacitor.

Inputs			Output
Clear	A	B	Q
0	X	X	0
1	0	↑	⎍
1	↓	1	⎍
↑	0	1	⎍

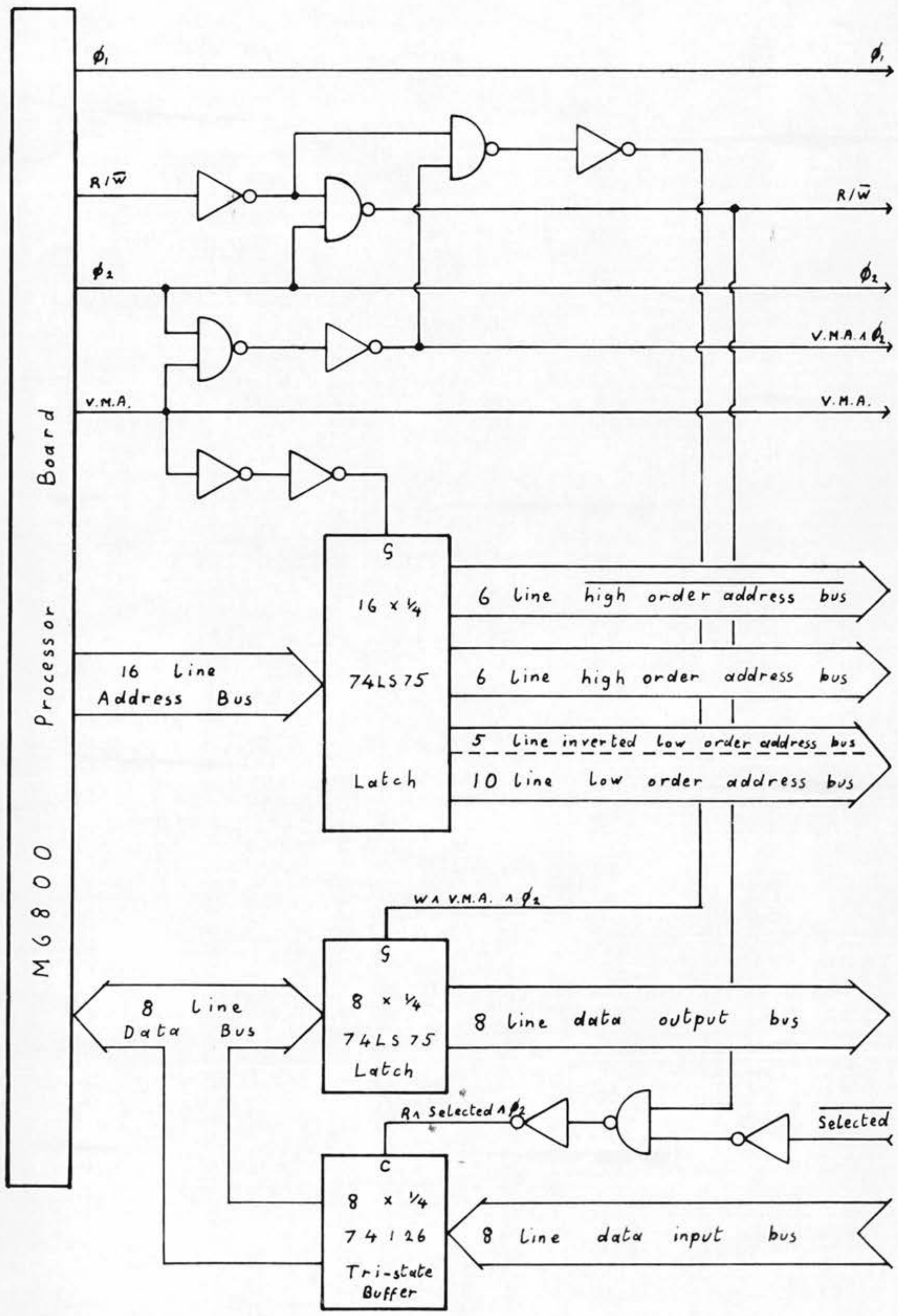


Modified Motorola Kit

MEK 6800 D1

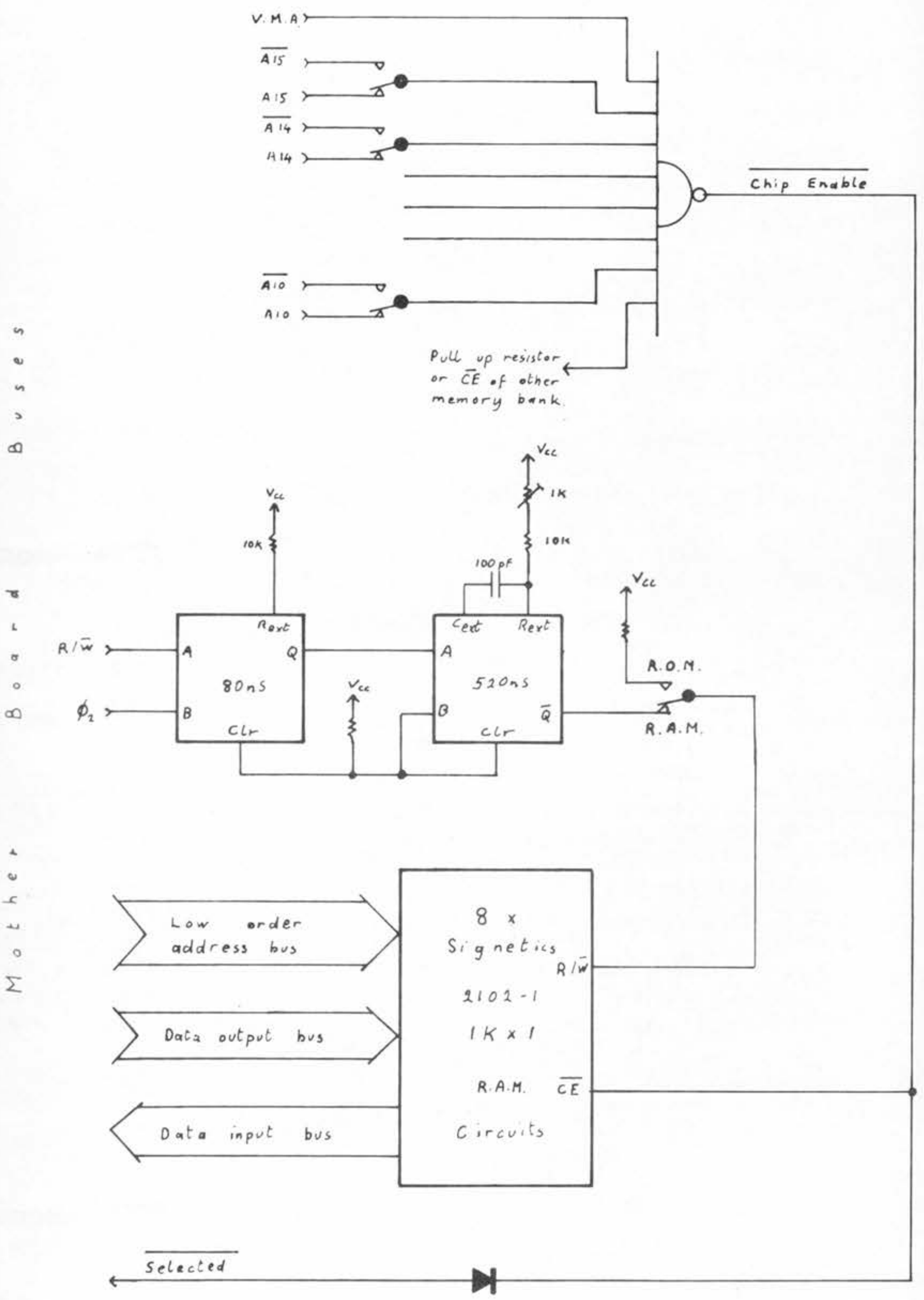


Appendix I(ii) MEK 6800 D1



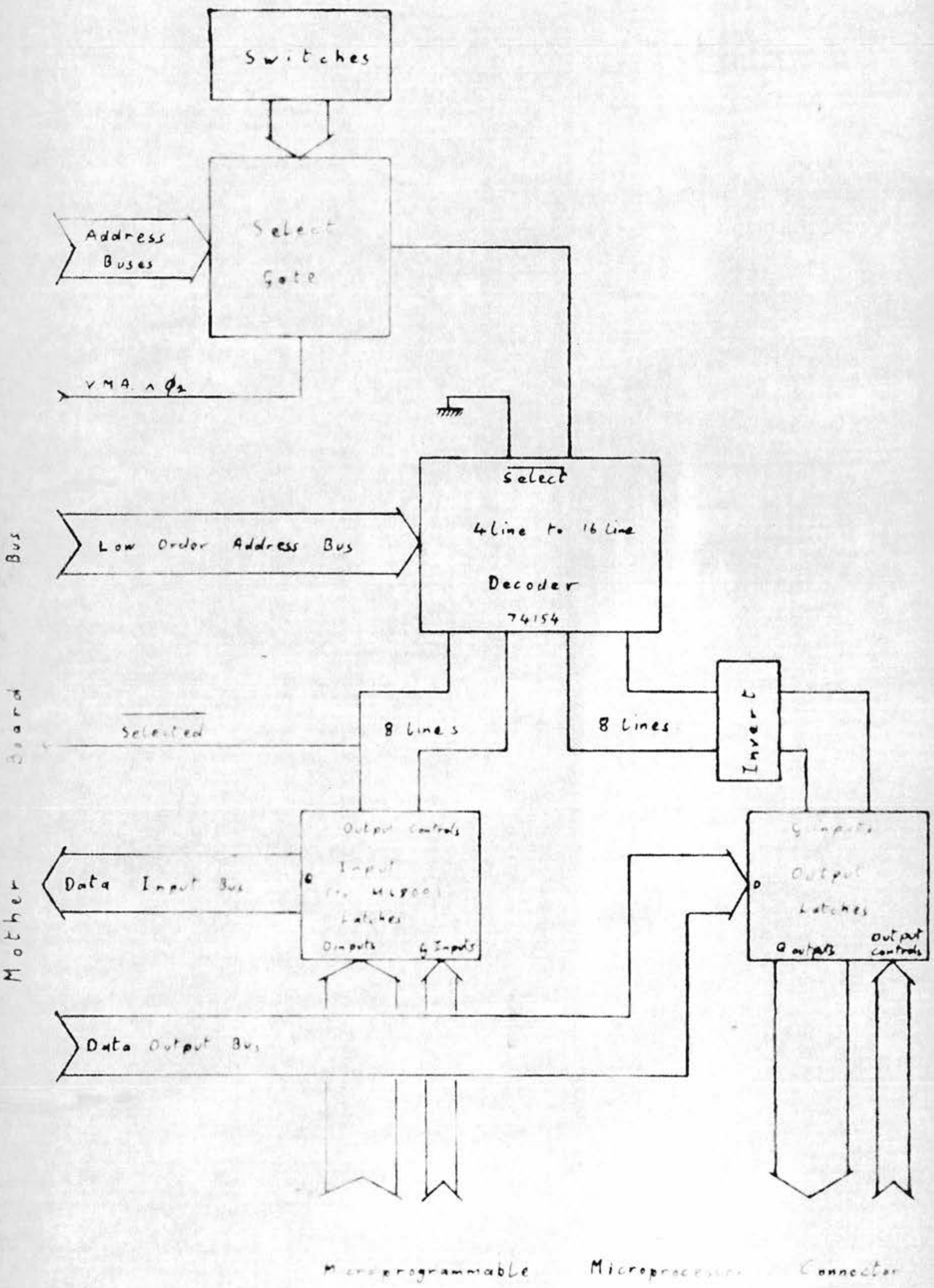
Appendix 1iii - Mother Board

Appendix I (iii) Mother Board



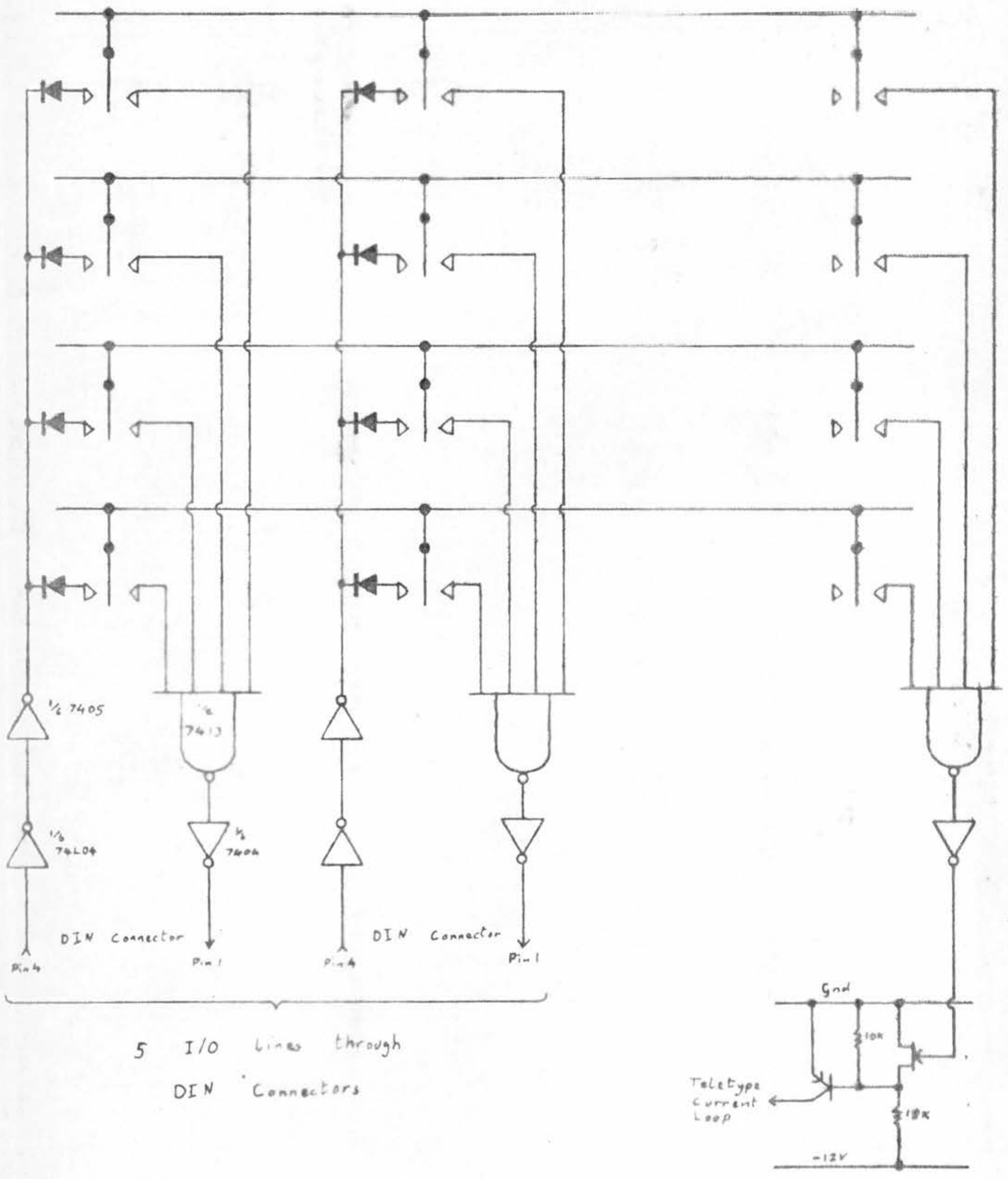
Appendix I(iv) – 1024 bit Random Access Memory

Appendix I(iv) R.A.M. Board



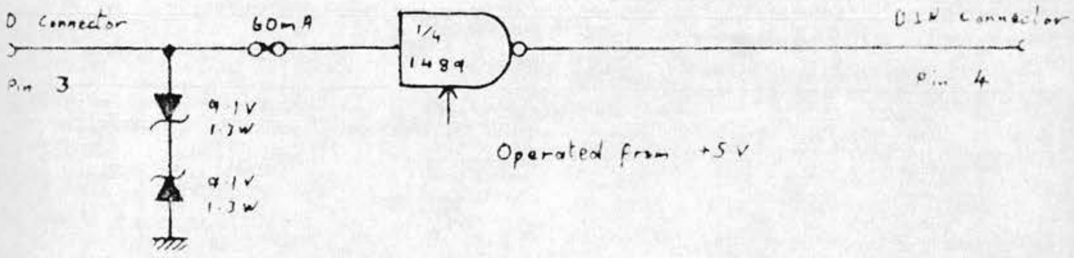
Appendix I(v) - Parallel Interface

Appendix I(v) Parallel Interface

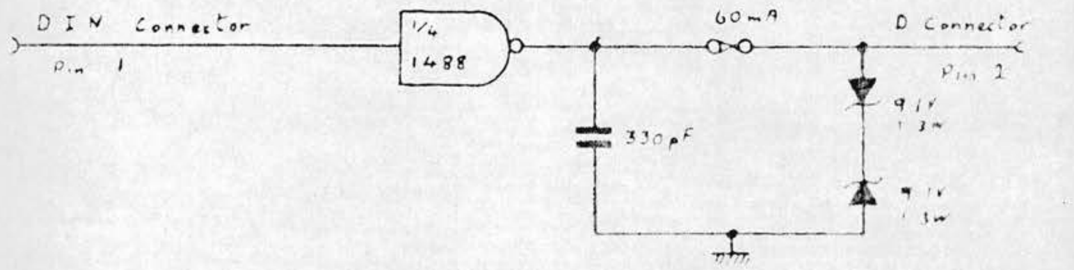


Appendix I (vi) - Exchange

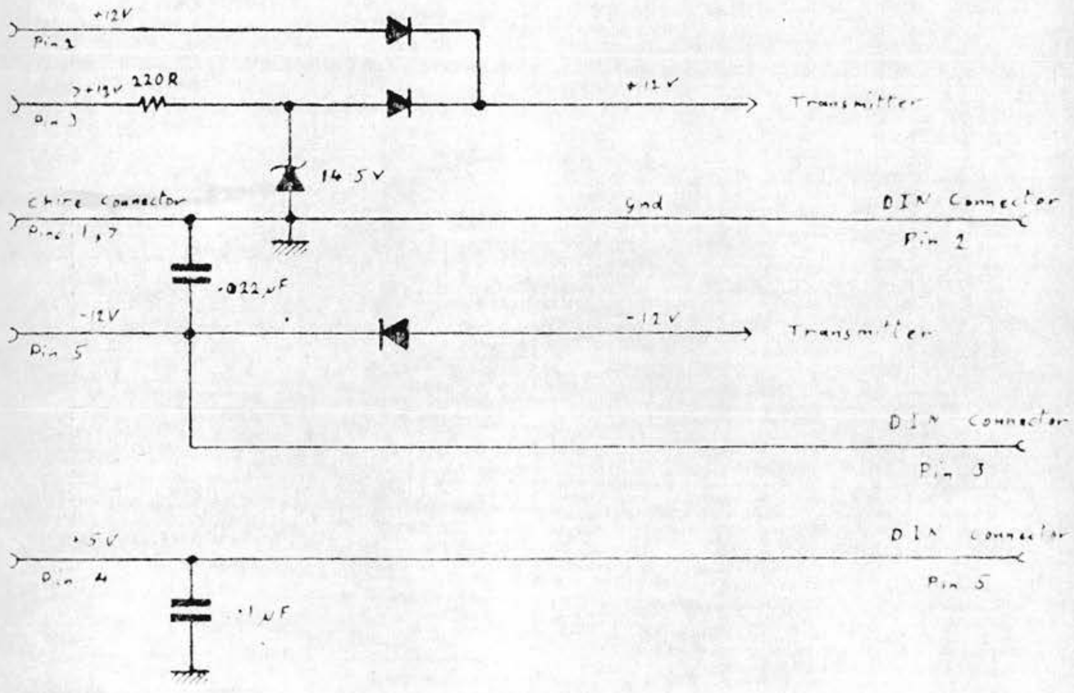
Appendix I(vi) Exchange



Receivers (2 off)



Transmitter (2 off)



Power

Appendix I (vii) V 24 Interface

Appendix 2(i)

Evaluation Kit MEK 6800 D1 Connections

Connector P1

The pin assignments are as described in the kit's guide[5], except as follows:

Pin L	Mikbug ROM enable (pin 11 of IC) - connect to A15 for normal operation, or to ground to disable the ROM.
Pins N,12	Keyway

Connector P2

Pin	Function
1	Gnd
2	CA1 3 PA0
4	PA2
5	PA4
6	PA6
7	PB0
8	PB2
9	PB4
10	PB6
11	CB1
12	Gnd
13	CTS
14	DCD
15	RX data
16	RX clock
17	TX clock
18	RTS
19	V_{cc}
20	$\overline{\text{Reset}}$
21	TX data
22	keyway
23	CB2
24	PB7
25	PB5
26	PB3
27	PB1
28	PA7
29	PA5
30	PA3
31	PA1
32	CA2

Connector P3

Pin	Function
1	Ground
4	110 Baud Select
6	300 Baud Select
8	110/300 baud select common
11	Reader Control (CB2)
12	Low / $\overline{\text{High}}$ data rate input
13	Serial Input (PA7)
14	Reset Input (Normally open circuit)
15	Serial Output (PA0)
16	Reset input (Normally ground)

Appendix 2(ii)

Mother Board Edge Connectors

Pin	Colour	Function	
1		V_{cc}	
2			
3	Black	A_0	
4	Brown	A_1	
5	Red	A_2	
6	Orange	A_3	
7	Yellow	A_4	
8	Green	A_5	Address Bus (Low Order)
9	Blue	A_6	
10	Violet	A_7	
11	Grey	A_8	
12	White	A_9	
13	Black	D_0	
14	Brown	D_1	
15	Red	D_2	
16	Orange	D_3	
17	Yellow	D_4	Data Input to Microprocessor from Memory
18	Green	D_5	
19	Blue	D_6	
20	Violet	D_7	
21	Black	D_0	
22	Brown	D_1	
23	Red	D_2	
24	Orange	D_3	
25	Yellow	D_4	Data Output from Microprocessor to Memory
26	Green	D_5	
27	Blue	D_6	
28	Violet	D_7	
29	Green	$\overline{A_5}$	
30	Blue	$\overline{A_6}$	
31	Violet	$\overline{A_7}$	Inverted Low Order Address Bus
32	Grey	$\overline{A_8}$	
33	White	$\overline{A_9}$	
34	White	$\overline{\text{Halt}}$	
35	Grey	Bus Available	
36	Violet	Three-state Control	
37	Blue	Clock $\emptyset_1$	
38	Green	Clock $\emptyset_2$	
39	Yellow	Valid Memory Address	
40	Orange	$\overline{\text{Reset}}$	
41	Red	$\overline{\text{N.M.I.}}$	
42	Brown	$\overline{\text{I.R.Q.}}$	
43		$\overline{\text{Selected}}$	
44		$\text{V.M.A.} \wedge \emptyset_2$	
45	Black	$\text{Read}/\overline{\text{Write}}$	

46	Black	A_{10},	47	$\overline{A10}$	
48	Brown	A_{11},	49	$\overline{A11}$	
50	Red	A_{12},	51	$\overline{A12}$	
52	Orange	A_{13},	53	$\overline{A13}$	High Order Address Buses
54		Keyway			
55	Yellow	A_{14},	56	$\overline{A14}$	
57	Green	A_{15},	58	$\overline{A15}$	
59					
60		Ground			

Appendix 2(iii)

Connections to Exchange etc.

DIN Connectors (Plugs, 5pin, 180^{o})

Pin 1 Receive TTL level serial data.
  2 Earth
  3 -12V (colour code clear)
  4 Transmit TTL level serial data.
  5 +5V (colour code yellow)

Chire Connector

Pin 1 Earth (Green/Yellow)
  2 +12V
  3 +12V (Brown)
Keyway
  4 +5V (Brown)
  5 -12V (Blue)
  6 Current loop to Teletype printer. (Blue)
  7 Earth (Green/Yellow)

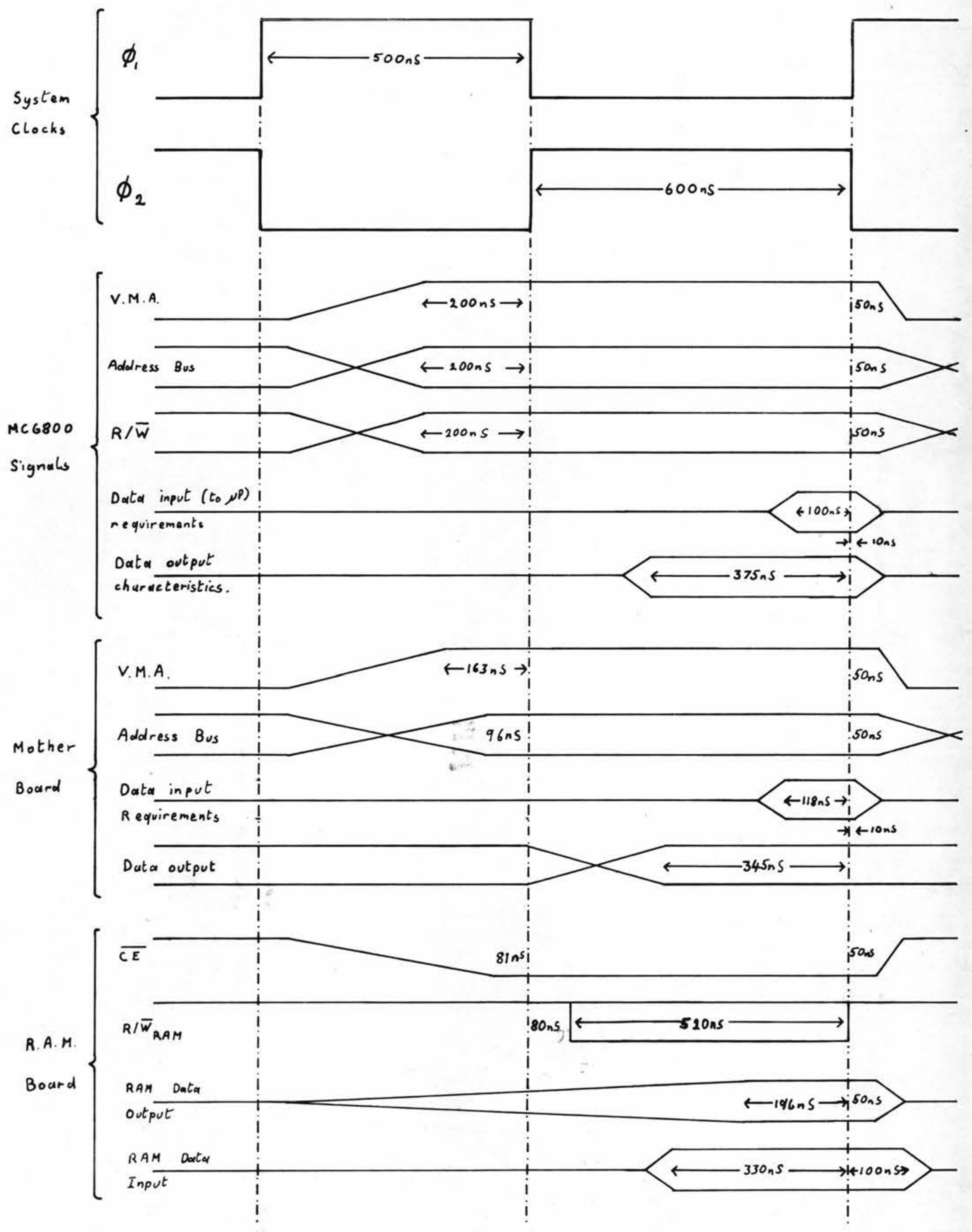
Miniature D-type Connector

Pin 2 Transmit
  3 Receive
  7 Ground

Below are listed the critical timing figures used to derive the timing diagrams opposite. The values underlined are taken as worst case - where no value is quoted in the references it has been necessary to impose a safe limit, e.g. 0nS for the minimum propagation delay through a gate.

		Minimum	Typical	Maximum	
MC6800	t_{AD} ($\phi_1\uparrow$ to Address, R/W, VMA)		220	$\underline{300}$	nS
	t_H ($\phi_1\uparrow$ to end of Address, R/W, VMA)	$\underline{50}$	75		nS
	t_{DSR} (Data setup to $\phi_2\downarrow$)	$\underline{100}$			nS
	t_H (Data input hold beyond $\phi_1\uparrow$)	$\underline{10}$			nS
	t_{DDW} (Data output delay from $\phi_2\uparrow$)		165	$\underline{225}$	nS
	t_H (Data output held beyond $\phi_2\downarrow$)	$\underline{10}$			nS
Gates	t_{PLH} (Normal TTL low to high prop. delay)	$\underline{0}$	11	$\underline{22}$	nS
	t_{PHL} (Normal TTL high to low prop. delay)	$\underline{0}$	7	$\underline{15}$	nS
Latches	t_{DQ} (Propagation delay, D input to Q output)	0		$\underline{30}$	nS
	t_{SQ} (Propagation delay, enable to Q output)	$\underline{0}$		$\underline{30}$	nS
	t_{PHL} (Tristate TTL high to low prop. delay)	0		$\underline{18}$	nS
Memory	t_A (Access time from address)			$\underline{500}$	nS
	t_{OH2} (Output hold time from $\overline{CE}$)	$\underline{0}$			nS
	t_{AW} (Address to write setup time)	$\underline{150}$			nS
	t_{WR} ($\uparrow$ write to $\uparrow\overline{CE}$ and addresses)	$\underline{50}$			nS
	t_{DW} (Data setup to R/$\overline{W}$ $\uparrow$)	$\underline{330}$			nS
	t_{DH} (Data hold time from R/$\overline{W}\uparrow$)	$\underline{100}$			nS

Sources: The TTL Data Book and Supplement, Texas Instruments Ltd.
Signetics mos and bipolar RAMS, Signetics Corporation.
M6800 data from reference 4.



Appendix 3 - Timing Diagram

Appendix 3 Timing Diagram

APPENDIX 4

Memory.test.c Listing

```
¢include "declaration"
¢define microprocessor_line "/dev/ttym"
  char c;
  int go, errors_detected;
  errors_detected = 0;
  go = 1;

¢include "stty.options"

  source = copen( argv[1], 'r' );
  prepare_load( wa );
  while( !ceof( source ) )cputc( cgetc(source), wa );
  pc = 0;
  sp = 0120102;
/*
  i.e. A042 in hexadecimal - the usual start address for the stack in the Mikbug.
*/
  restore_registers( wa );
  read_registers( ra );
  printf( "Registers contain: A = %x, B = %x, PC = %x, X = %x.\n", a, b, pc, x );
  wait( ra );
  cputc( 'G', wa );
  while( go )
  {
    read_registers( ra );
    if( x == 0134000 )
    {
      printf( "\n%d errors detected.\n", errors_detected );
      go = 0;
      continue; };
    printf( "Error detected: Address %x,  %x instead of %x.\n", x, b, a );
    errors_detected =+ 1;
    if( errors_detected % 3 == 0 )
    {
      printf( "\nNumber of errors detected = %d.\nType t to terminate, c to continue
              or n to continue with the next byte.", errors_detected );
      {
        switch( getchar() )
        {
          case 'n': a = 0;
                    x =+ 1;
                    set( x, 0 );
                    printf( "\nNext byte\n" );
                    break;
          case 't': go = 0;
                    printf( "\nTerminating\n" );
                    break;
          default:
          case 'c': a = b; };
        getchar(); };
      else a = b;
    pc =+ 1;
    restore_registers( wa );
    wait( ra )
    cputc( 'G', wa ); };
  for( ;; c = cgetc( ra ) )putchar( c ); };

¢include "procedures"
```

Appendix 4 Memory.test.c Listing

References

1. P.D.P. 11 Peripherals Handbook, Digital Equipment Corporation.

2. C Reference Manual, Ritchie D. M., Bell Telephone Laboratories Internal Publication.

3. The UNIX Time-Sharing System, Ritchie D. M. and Thompson K., C.A.C.M. Vol 17/7 p365.

4. M6800 Systems Reference & Data Sheets, Motorola Semiconductor Products Inc. or M6800 Microcomputer System Design Data for more up to date information.

5. User's and Assembly guide for the Motorola Microcomputer Distributor Kit MEK 6800 D1, Motorola Semiconductor Products Inc.

6. Engineering Note 100 - MCM6830L7 Mikbug/Minibug ROM, Motorola Semiconductor Products Inc.

7. Technical Guide No. 2, Post Office Datel Services - General requirements for Data Terminal Equipment, Post Office Telecommunications Services.

8. The Unix Motorola M6800 Assembler Reference Manual, Jon Rowson, Computer Systems Laboratory, Queen Mary College. (Available in NROFF text processor format under /usr/david/m6800/ds.refmanual.n)

9. MAS - Motorola M6800 Assembler. (Available in NROFF format under /usr/david/m6800/ds.mas.1)

10. Unix Motorola M6800 Communication Routines, David Salmon, University of St. Andrews. (Available in NROFF format)

Acknowledgements

Throught this project the University of St. Andrews Department of Physics have been most co-operative lending equipment, workshop space and giving advice. Particular mention should be made of Mr. F. C. Evans and Mr. M. Robertson.

Great appreciation is recorded of the help and encouragement given by the members of the Department of Computational Science and the Computer Laboratory. Mr. M. Weatherill, the Author's supervisor, deserves recognition by name.

The project was financed with the help of a grant from the Science Research Council.

This was typed using a Fortronic F100 word processor, the use of which is acknowledged.