EXPANDED INPUT / OUTPUT CARD FOR THE AXIOM CME11E9-EVBU DEVELOPMENT BOARD

A Report Presented to the
Faculty of the Computer Engineering Technology Program
University of Southern Mississippi

In Partial Fulfillment of the Requirements for
CET 401

By
Dan Kohn
May 8, 1999

Table of Contents

I. ABSTRACT **1**

II. INTRODUCTION **2**

III. CIRCUIT DESIGN AND OPERATION **3**

A. System Overview 3
B. Addressing 5
1. CME11E9-EVBU Peripheral Addressing *5*
2. Expanded IO Card Addressing *7*
a. Hardware 7
b. Testing 8
C. Expanded Input / Output 10
1. Input Ports *10*
a. Hardware 10
b. Testing 10
c. Software 11
2. Output Port *13*
a. Hardware 13
b. Testing 13
c. Software 14
3. Motor Control Port *16*
a. Modifications from Initial Design 16
i. Direct Output Control 16
ii. Motor Clock Problem and Solution 17
iii. Motor Output Circuit Modification 18
b. Jumper Settings for the Motor Control Port 19
d. Stepper Motor Control 19
i. Hardware 19
ii. Software 21
iii. Testing 24
c. DC Motor Control 27
i. Hardware 27
ii. Software 27
iii. Testing 28
4. Analog Output *31*
a. Modifications from initial design 31
b. Hardware 32
c. Testing Analog Output Addressing 34
d. Software 34
e. Testing Analog Output 35
D. Small C Software Library 36
E. Test Software 38
F. CME-PC Visual Basic Interface Program 38

REFERENCES **39**

APPENDICES **40**

Table of Figures

FIGURE 1. SYSTEM BLOCK DIAGRAM........3
FIGURE 2. CME11E9-EVBU PERIPHERAL ADDRESSING CIRCUIT........5
FIGURE 3. EXPANDED I/O CARD ADDRESSING DECODER........7
FIGURE 4. INPUT ADDRESSING SAMPLE WAVEFORM........8
FIGURE 5. OUTPUT ADDRESSING SAMPLE WAVEFORM........9
FIGURE 6. INPUT PORT........10
FIGURE 7. INPUT PORT TEST CIRCUIT........11
FIGURE 8. INPUT PORT EXAMPLE........12
FIGURE 9. OUTPUT PORT........13
FIGURE 10. OUTPUT PORT TEST CIRCUIT........14
FIGURE 11. OUTPUT PORT EXAMPLE........15
FIGURE 12. MOTOR CONTROL PORT........16
FIGURE 13. MOTOR CLOCK NOISE EXAMPLE........17
FIGURE 14. MOTOR CLOCK WITH SCHMITT TRIGGER........18
FIGURE 15. CLOCK FILTER (SCHMITT TRIGGER CIRCUIT)........18
FIGURE 16. MOTOR CONTROL PORT JUMPER SETTINGS........19
FIGURE 17. STEPPER MOTOR CONFIGURATION........20
FIGURE 18. OUTPUT COMPARE ISR........22
FIGURE 19. SAMPLE STEPPER MOTOR WAVEFORMS........25
FIGURE 20. SAMPLE STEPPER MOTOR CLOCK........26
FIGURE 21. DC MOTOR CONFIGURATION........27
FIGURE 22. PWM SIGNAL AT 35%........30
FIGURE 23. PWM SIGNAL AT 75%........30
FIGURE 24. D/A ADDRESSING TIMING ERROR........31
FIGURE 25. CORRECTED D TO A ADDRESSING........32
FIGURE 26. D/A CONVERTER........33
FIGURE 27. D TO A ADDRESSING SAMPLE WAVEFORM........34
FIGURE 28. FLOWCHART FOR DEBUG()........37

I. Abstract

With the continuing encroachment of computers into all aspects of day-to-day life, the need for interfacing between the real world and the digital world is on the increase. This paper describes the circuitry and software for additional input/output capabilities for the CME11E9-EVBU Development Board. When combined, the system created is a very powerful controller for a wide variety of applications including industrial control, robotics, data acquisition, and home automation.

II. Introduction

The Expanded Input / Output (I/O) Card is designed to allow the end user the ability to quickly connect various devices to the Axiom CME11E9-EVBU Development Board by providing the necessary interfacing hardware for a wide range of applications. The Expanded I/O Card can handle up to sixteen additional CMOS Level inputs, 4 additional CMOS Level outputs, four transistor switched outputs, four analog outputs, and two motor control outputs (each capable of controlling either a DC motor or Unipolar Stepper Motor). When these additional I/O capabilities are combined with the 68HC11's on board Input Capture , Output Compare, and Analog to Digital Converter (A/D), there are very few devices that cannot be interfaced to and controlled by this system.

III. Circuit Design and Operation

A. System Overview

The overall system is comprised of two main parts: the CME11E9-EBVU Development Board and the Expanded I/O Card. The CME11E9-EBVU Development Board consists of a 68HC11 Microcontroller Unit – MCU, memory for program storage, addressing hardware for peripheral hardware, and an RS-232 port (for programming and data exchange). The Expanded I/O Card receives addressing information and data from the Development Board and uses those signals to control the additional I/O capabilities. A block diagram of the overall system can be found in Figure 1.

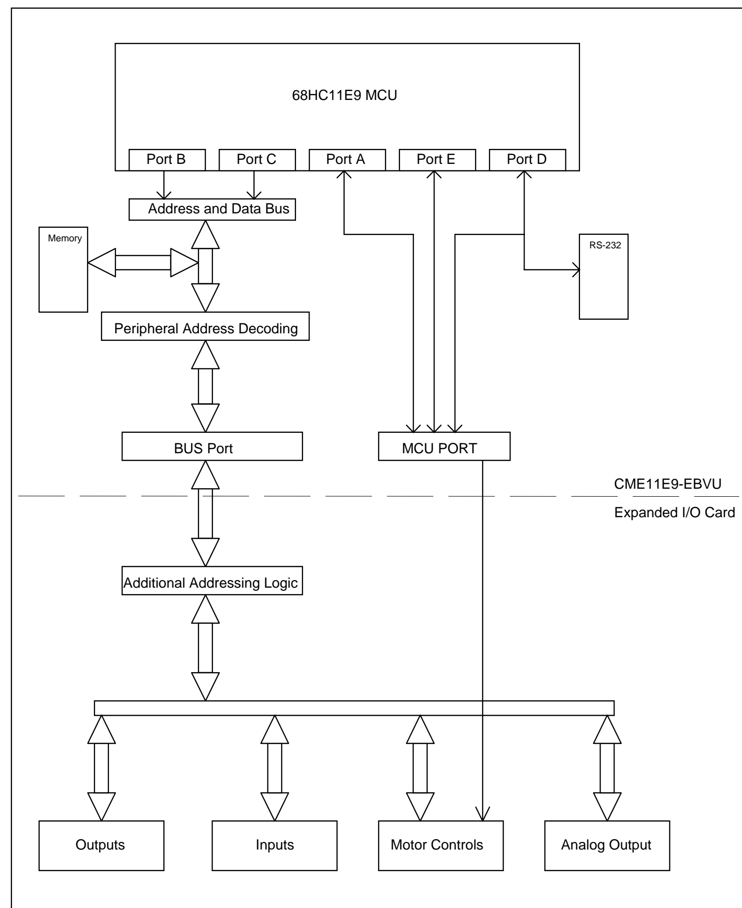


Figure 1. System Block Diagram

Without the Expanded I/O Card, the CME11E9-EVBU Development Board has limited I/O capabilities. With the 68HC11 on the CME11E9-EVBU in expanded mode [1], parallel Ports B and C are used for addressing and are unavailable. Port D is partially used for RS-232 communications so only four pins of that port are available. This leaves only 50% of the 68HC11's I/O capabilities available for use. Out of these remaining I/O pins, many have special functions, are unidirectional, or are CMOS level outputs, putting further limits on the applications of the Development Board. Lastly, the Development Board has no outputs capable of handling higher current devices (motors, relays, lights ect) requiring interfacing circuitry for the majority of applications.

The Expanded I/O Card solves these problems. With the addition of sixteen CMOS level inputs, four CMOS Level outputs, and four high voltage/current outputs, the Expanded I/O Card can be connected to any number of devices. The inclusion of the two Motor control ports increases the usefulness of the Expanded I/O Card while reducing software overhead by allowing hardware to handle the majority of the motor controls without extensive processor supervision. The inclusion of a Digital to Analog converter adds additional functionality unavailable without the Expanded I/O Card. Detailed specifications and full schematics for the Expanded I/O Card are available in Appendix B.

B. Addressing

1. CME11E9-EVBU Peripheral Addressing

Before discussing the I/O circuitry, it is important to develop and understanding of the method used for addressing for each of the additional ports. The responsibility for addressing these ports is shared by the CME11E9-EVBU Development Board and the Expanded I/O Card.

The CME11E9-EVBU Development Board (see Appendix A for full schematic) handles preliminary addressing for the Expanded I/O Card. U2 and U10 as seen in Figure 2 handle this task. U2 is a Programmable Array Logic (PAL) IC. This 16V8 is programmed with the necessary logic for all the addressing needs of the

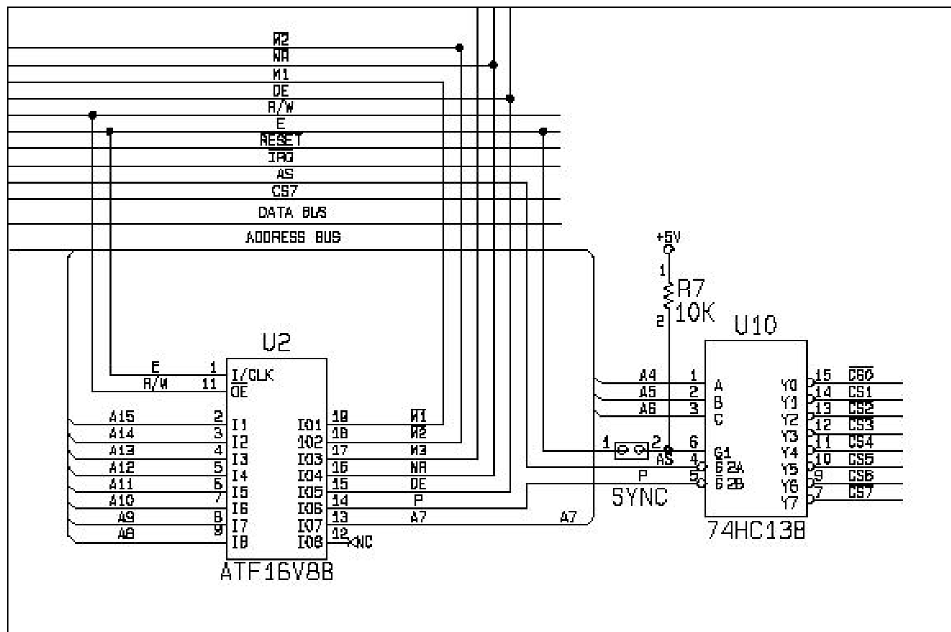


Figure 2. CME11E9-EVBU Peripheral Addressing Circuit [2]

CME11E9-EVBU Development Board as well as the Peripheral Addressing Area. Only the P output is used for the Peripheral Addressing. The other outputs are used for memory addressing and will not be covered in this report. The P line goes LOW when the address bus contains an address in the Peripheral Area (addresses $B5N0 to $B5FF). The logic for the P is:

(Equation 1)

$$P = NOT(A15 \,\&\, \overline{A14} \,\&\, A13 \,\&\, A12 \,\&\, \overline{A11} \,\&\, A10 \,\&\, \overline{A9} \,\&\, A8 \,\&\, A7 \,\&\, E)$$

where: A0..A15 are address lines

E is the bus clock

P is the output (active low)

The Development board then uses this signal, along with others, to generate 8 lines (CS0..CS7) indicating which subdivision of the Peripheral Address Area is being addressed. These subdivisions are shown below:

Table I. Subdivisions of Peripheral Address Area

CS0 – $B580 - $B58F	CS4 – $B5C0 - $B5CF
CS1 – $B590 - $B59F	CS5 – $B5D0 - $B5DF
CS2 – $B5A0 - $B5AF	CS6 – $B5E0 - $B5EF
CS3 – $B5B0 - $B5BF	CS7 - $B5F0 - $B5FF*

*note: $B5F0 and $B5F1 are reserved for the LCD Port

The determination of the subdivision being addressed is the responsibility of U10 in Figure 2. This inverting 3-to-8 line decoder/demultiplexer (74HC138) receives P (Peripheral Area Address Selected), A4..A6, AS (Address Strobe), and E (Bus Clock) and generates $\overline{CS0}$ .. $\overline{CS7}$ indicating the subdivision being addressed. The logic is shown in the Truth Table below:

Table II. 74HC138 Truth Table

A (A4)	B (A5)	C (A6)	G1 (E)	$\overline{G2A}$ (AS)	$\overline{G2B}$ (P)	OUTPUT*
0	0	0	1	0	0	$\overline{CS0}$
1	0	0	1	0	0	$\overline{CS1}$
0	1	0	1	0	0	$\overline{CS2}$
1	1	0	1	0	0	$\overline{CS3}$
0	0	1	1	0	0	$\overline{CS4}$
1	0	1	1	0	0	$\overline{CS5}$
0	1	1	1	0	0	$\overline{CS6}$
1	1	1	1	0	0	$\overline{CS7}$

* - all outputs and possible states not shown are 1's

The $\overline{CS0}$ .. $\overline{CS7}$ lines, along with other lines are then made available via connector J1 (Bus_Port) on the CME11E9-EVBU Development Board.

2. Expanded IO Card Addressing

a. Hardware

The Expanded I/O Card connects to the Bus_Port and uses A0, A1, A2, A3 , $\overline{RW}$, and $\overline{CSn}$ to complete the addressing. As shown in Figure 3, a jumper is used to select the Peripheral Address line ($\overline{CS0}$ .. $\overline{CS7}$) to be used.

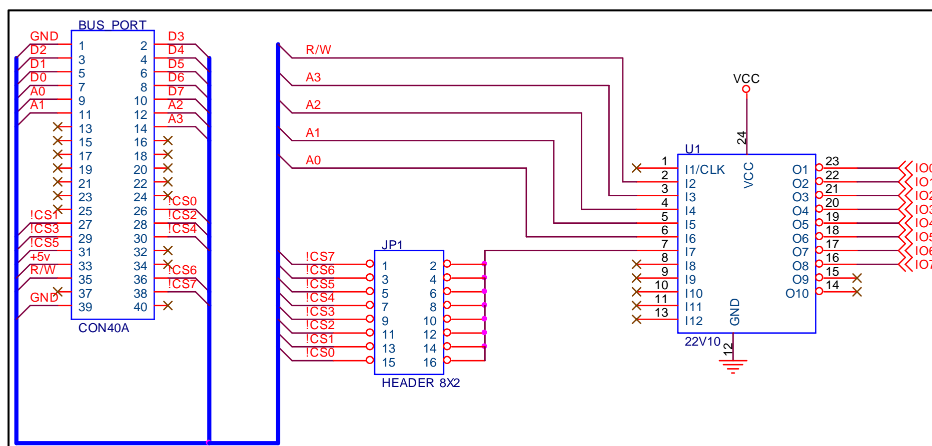


Figure 3. Expanded I/O Card Addressing Decoder

This was done to allow the Expanded I/O Card to be connected to Development Boards that already has some of the peripheral addresses in use. The selected peripheral address line ($\overline{CSn}$) (where n is the CS line selected by the jumper JP1) along with A0..A3 and $\overline{RW}$ are then used as inputs to U1 in Figure 3. This PAL22V10 is programmed to produce the necessary outputs required for each I/O sub-circuit. Table III contains the truth table used to create the combinational logic.

Table III. Addressing PAL Truth Table

ADDR	$\overline{CSn}$	$\overline{RW}$	A3	A2	A1	A0	Output	Port Type
\$B5n0	0	1	0	0	0	0	$\overline{IO0}$	Input
\$B5n1	0	1	0	0	0	1	$\overline{IO1}$	Input
\$B5n2	0	0	0	0	1	0	$\overline{IO2}$	Output
\$B5n3	0	0	0	0	1	1	$\overline{IO3}$	Motor

$B5n4	0	0	0	1	0	0	special	D/A – OUTA
$B5n5	0	0	0	1	0	1	special	D/A – OUTB
$B5n6	0	0	0	1	1	0	special	D/A – OUTC
$B5n7	0	0	0	1	1	1	special	D/A - OUTD
$B5n8	0	0	1	0	0	0	$\overline{IO4}$	$\overline{LDAC}$

$B5n4..$B5n7 generate special outputs needed to control the multiplexed data inputs of the D/A, and will therefore be covered in detail with that device. The PAL programming file along with further documentation is available in Appendix C.

b. Testing

The procedure to test the addressing circuit can be broken up into three procedures. Since the test code is dependent upon the JP1, 'n' will replace the hex digit selected via this jumper as per Table I.

The first procedure tests the input port addressing. For the first Input Port (IO0), enter the following code into the CME11E9 via the Buffalo Monitor at location 2400:

```
LDAA   B5n0
JMP    2400
```

Then run the code. Place a scope on Pin 23 of U1 (IO0). The output should be as follows:

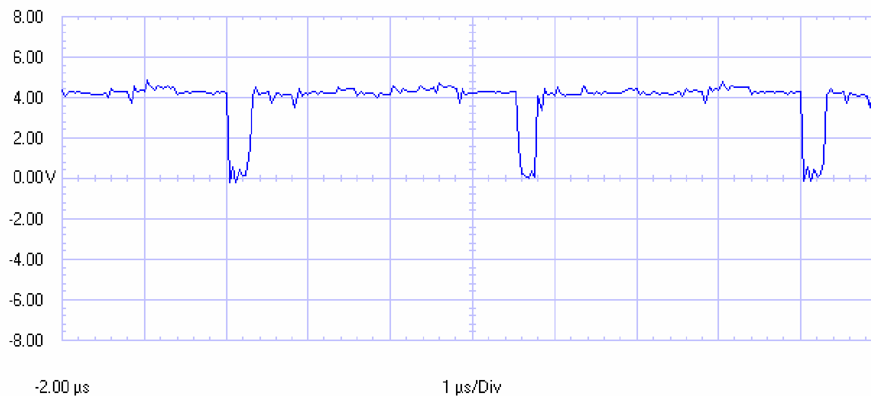


Figure 4. Input Addressing Sample Waveform

This waveform verifies that when B5n0 is addressed the IO0 pin goes low and sends the correct signal to the input circuit. At this point all the other pins were check to verify that this signal does not appear on any of the other output pins (IO1..IO8)

The procedure above was repeated for IO1, changing address on the first line of assembly language code changed to B5n1 and moving the scope to pin 22 of U1 and again verifying the waveform and no bleed over to other IO lines.

The next procedure tests the output port addressing. Very similar to the input port test, the code entered is as follows:

```
STAA    B5n2
JMP     2400
```

Then run the code. Place a scope on Pin 21 of U1 (IO2). The output should be as follows:

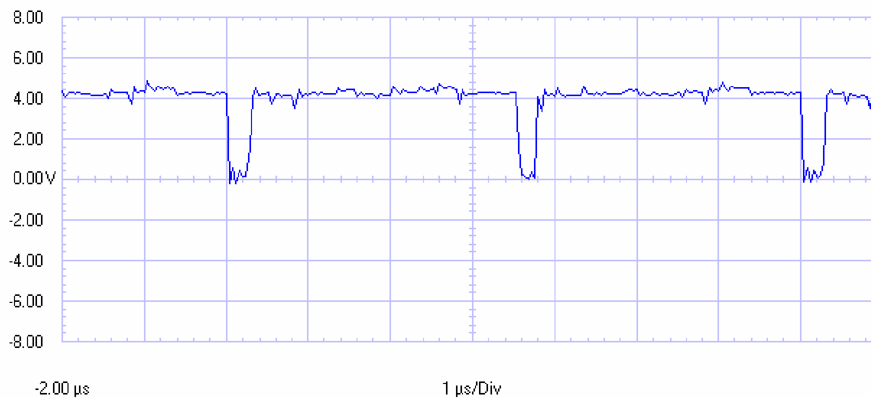


Figure 5. Output Addressing Sample Waveform

This waveform verifies that when B5n2 is addressed the IO2 pin goes low and sends the correct signal to the output circuit. At this point all the other pins were check to verify that this signal does not appear on any of the other output pins (IO0..IO1 and IO3..IO8)

The procedure above was repeated for the motor control address line (IO3), changing address on the first line of assembly language code changed to B5n3and moving the scope to pin 20 of U1 and again verifying the waveform and no bleed over to other IO lines.

The procedure for testing the final addressing lines will be discussed with the D/A converter in section III part 4c.

C. Expanded Input / Output

1. Input Ports

a. Hardware

The Expanded I/O Card is equipped with two input ports. These ports use a 74HC244 Octal

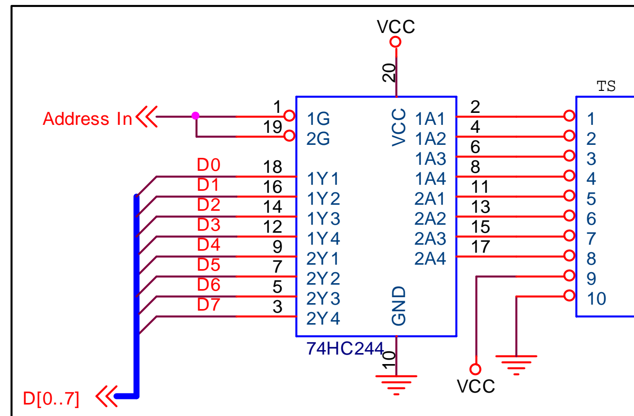


Figure 6. Input Port

buffer / line driver IC to transfer the data on the input lines (TS in Figure 6) onto the data bus when the Address line (IO0 or IO1 in Figure 3) goes low. This makes the data on TS directly accessible to the software by the use of memory fetching commands.

b. Testing

To test the input ports, the following test circuit can be used to connect to TS in Figure 6 to supply the proper input voltage levels. This simple circuit switches +Vcc to the input pins when the dip switches are on and ground when they are off (if Vcc and Ground are reversed, so are the outputs).

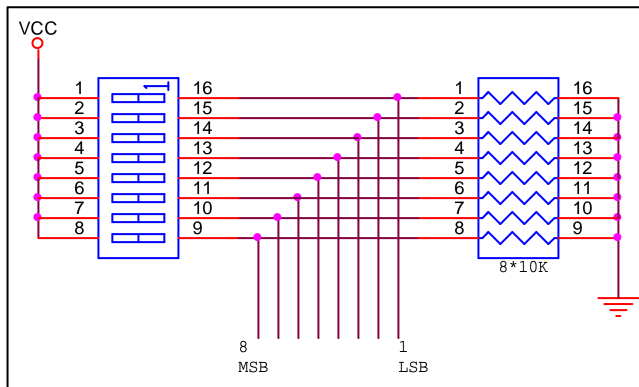


Figure 7. Input Port Test Circuit

With the test circuit connected, the input ports can be tested by using the Buffalo Monitor's display (D) command. Simply set the dip switch to any value and then read the port by typing d B5n0 for input #0 or d B5n1 for input #1. The following section shows how to convert the value displayed. To verify the input port received the correct value.

c. Software

To read in data from either of the Input Ports on the Expanded I/O Card, the programmer must read from the proper address. As shown in Table III, the address for the two input ports are $B5n0 and $B5n1. Using Small C's peekb() function, or assembly's LDA instruction, the port can be read. The value returned can be evaluated by breaking the value to it's binary equivalent to obtain the status of each input line. An example is as follows:

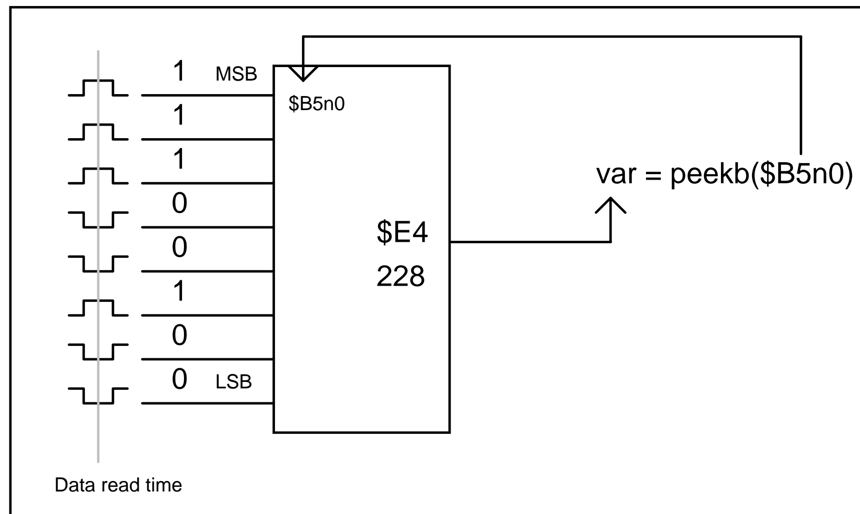


Figure 8. Input Port Example

As seen in Figure 8, the input port at $B5n0 is read using peek($B5n0). The value $E4 (228 dec) is read from the hardware and the data is placed into the variable "var". This variable can then be evaluated further to determine the status of each input line using any number of techniques.

It should also be noted that writing to an input port address, by use of pokeb() or STA, will have no effect on that address.

2. Output Port

a. Hardware

Figure 9 shows the configuration of the output port. The 74HC374 is an Octal D-type flip-flop; positive edge-trigger; 3-state. When triggered via the IO2 line from U1 (PALCE22V10 used for addressing), the information on the data bus will be latched onto the outputs (Q0..Q7).

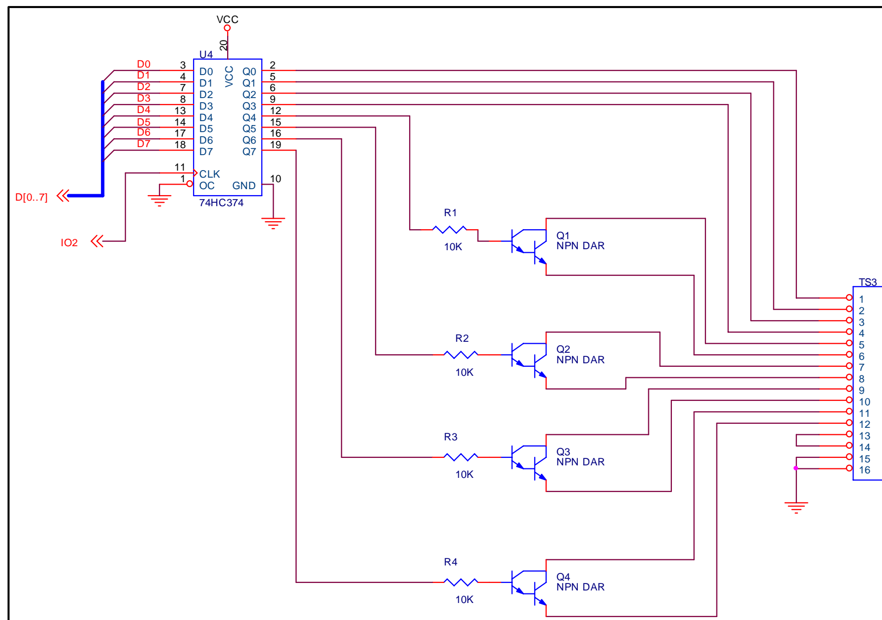


Figure 9. Output Port

To allow for higher voltage and/or current devices to be interfaced to the board, output Q4..Q7 are connected to TIP120s. These are Medium-Power Complementary Silicon Transistors arranged in a Darlington configuration. This IC was chosen for its high input impedance, high current gain, and low output impedance. The TIP120's collectors and emitters are brought directly to a terminal strip so that any connection configuration can be achieved.

b. Testing

The best way to test this circuit is to connect up the following test circuit to the output port's terminal strip.

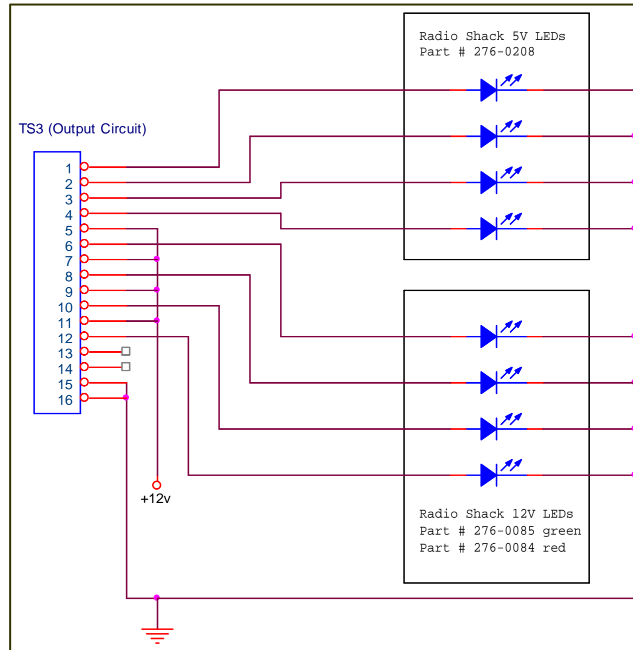


Figure 10. Output Port Test Circuit

Once connected, Buffalo can be used to test the output port. Using the Memory Modify (MM) to send out values to the output port (B5n2). The light pattern should change to match the value sent. It should be noted that the Buffalo Monitor will say "-rom" when memory modify is used in this way. Since the output port is one way (ie the value changes the output, but cannot be read back from the port) the check performed by Buffalo to verify the memory address changed will fail and return this message. You will notice, however, that the status lights did change to match the given value.

c. Software

Data can be placed on the Output Port of the Expanded I/O Card by use of Small C's pokeb() function or assembly's STA instruction. In the example in Figure11,

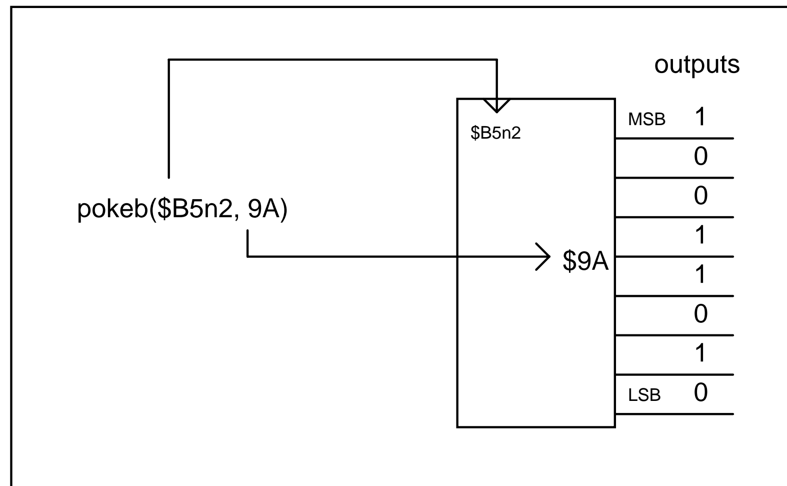


Figure 11. Output Port Example

the output port ($B5n2) has a HEX 9A written to it, the individual output lines then become a 1 (TRUE, +5v) or a 0 (False, 0v) based on the binary equivalent .

It should be noted that reading this port using LDA or peek() functions will **not** result in the ports current output. Therefore it is imperative that the current output be stored by some other means.

3. Motor Control Port

The motor control port (Figure 12) uses the 74HC374 for the same application as in the output port discussed above. This provides a latched output to the two PALCE22V10s. These Programmable Array Logic IC (PAL) are programmed to allow the CME11E9-EVBU Development Board to control either a DC motor or Unipolar Stepper motor with minimal software overhead. Full documentation of the DC/Stepper motor control PAL is available in Appendix D.

Figure 12. Motor Control Port

Since the 74HC374 is being used for the same application as the output port, the same testing procedure can be used to check the output of U5 and is therefor not repeated in this section.

a. Modifications from Initial Design

i. Direct Output Control

An additional feature of the motor control port has been added since the initial design. This feature gives the option of directly controlling the output directly from software. This feature adds flexibility to the Expanded I/O Card with only a minor change to the PLD code and the replacement of the mode selector switch with jumpers. The jumper settings for each mode are discussed below.

ii. Motor Clock Problem and Solution

During proto-board testing of the circuit it was discovered that the PLD's were very susceptible to clock noise. The problem caused some problems with the DC Motors, but was very pronounced during stepper motor operation. This is best illustrated by the following waveform:

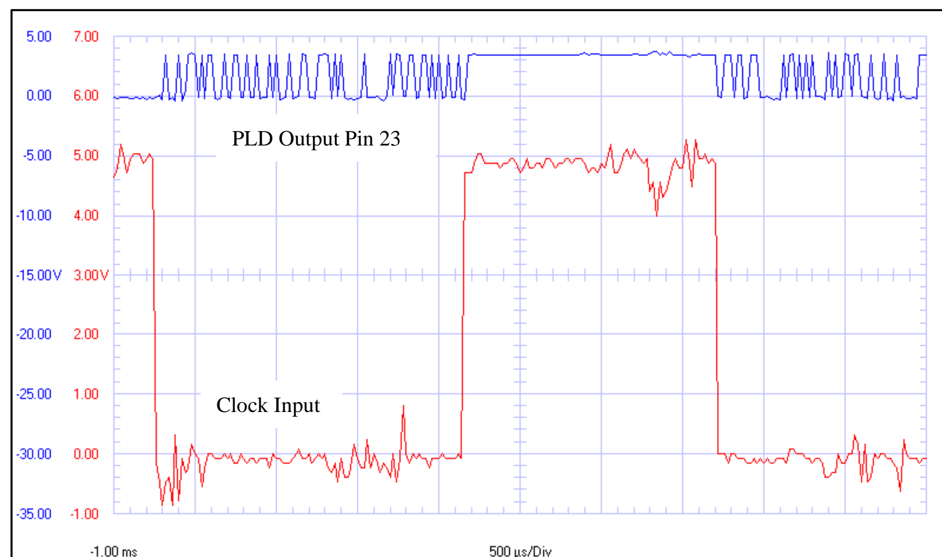


Figure 13. Motor Clock Noise Example

The solution to this problem was to add a 74HC14 Hex Schmitt-Trigger Inverter IC (Appendix E) into the circuit between the clock output from the CME11E9 Board and the Motor Control Pal. Even though only one Schmitt Trigger was needed to remove the noise from the clock signal, two were used for each channel to re-invert the signal to its original orientation. Below is an example of the clock after the Schmitt-Triggers were added:

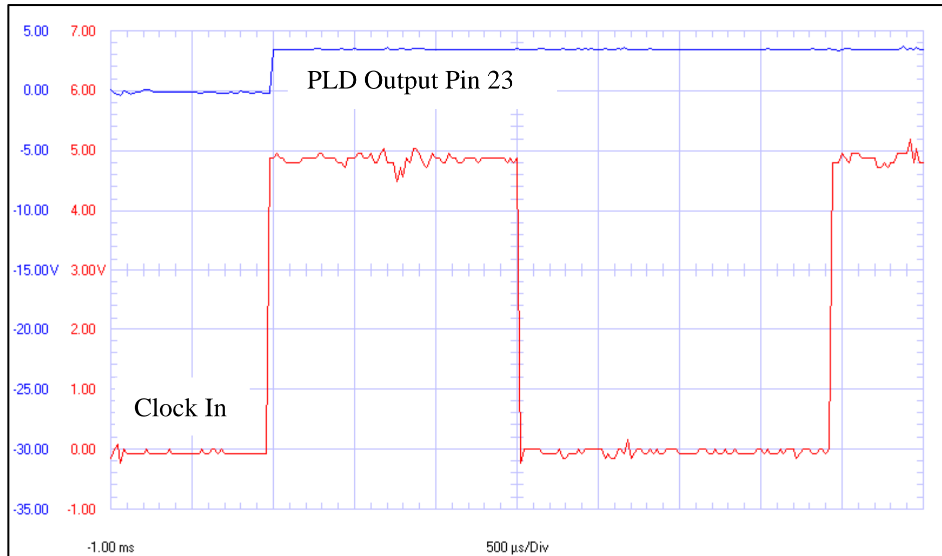


Figure 14. Motor Clock With Schmitt Trigger

The additional circuitry is shown below:

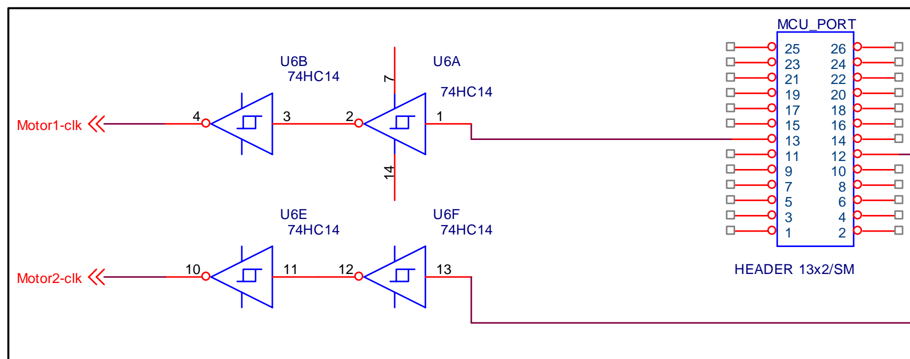


Figure 15. Clock Filter (Schmitt Trigger Circuit)

iii. Motor Output Circuit Modification

In the initial design of the Expanded IO Card, there was an error in the output section of the motor control port that would not allow the TIP120 Darlington Pairs in the upper leg of the DC motor H-Bridge to go into saturation. This was because the load of the Darlington Pairs are on the emitter and the base was only being supplied with +5v, instead of the +12v needed for the operation of the transistor with the load on the emitter. This problem was corrected with the edition o f the 2N2222A transistors (Appendix F) and

related components for each motor control ports (Q13 - Q16 in Figure 12). Since the inclusion of these transistor switching circuits inverts the output signal, a NOT was added in the PAL logic for the associated pins.

Lastly, the 5th output transistor (used in the initial design for PWM only was removed and the PAL logic was changed to allow the PWM signal to be applied to one of the remaining four transistor outputs.

For the code for the motor control PAL refer to Appendix D.

b. Jumper Settings for the Motor Control Port

The following diagram shows the jumper settings for the three modes of operation for the Motor Control Ports. Each port can independently be set for Stepper Motor Control, DC Motor Control or Direct Control. Jumpers JP2 and JP3 should be set as per the following diagram for the required function:

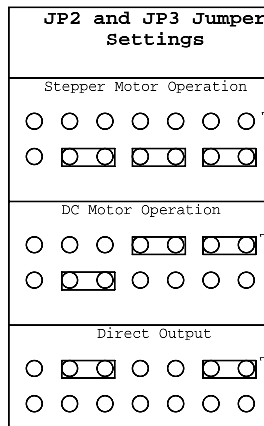


Figure 16. Motor Control Port Jumper Settings

d. Stepper Motor Control

i. Hardware

For Unipolar Stepper Motor control, the jumpers should be set according to Figure 16. The jumpers set the following pins on the PAL22V10:

- Pin 1 is connected to the 68HC11 clock pulse (via the Schmitt Trigger Filter) that is generated by the MCU's Output Compare (OC) facility and disables the DC motor logic.
- Pin 3 of the respective PAL to a LOW state - selecting stepper motor sequential logic.

- Pin 4 is set LOW to indicate motor control (not allowing direct control of the output pins).

The stepper motor(s) will be connected to TS4 and/or TS5 as per Figure 17

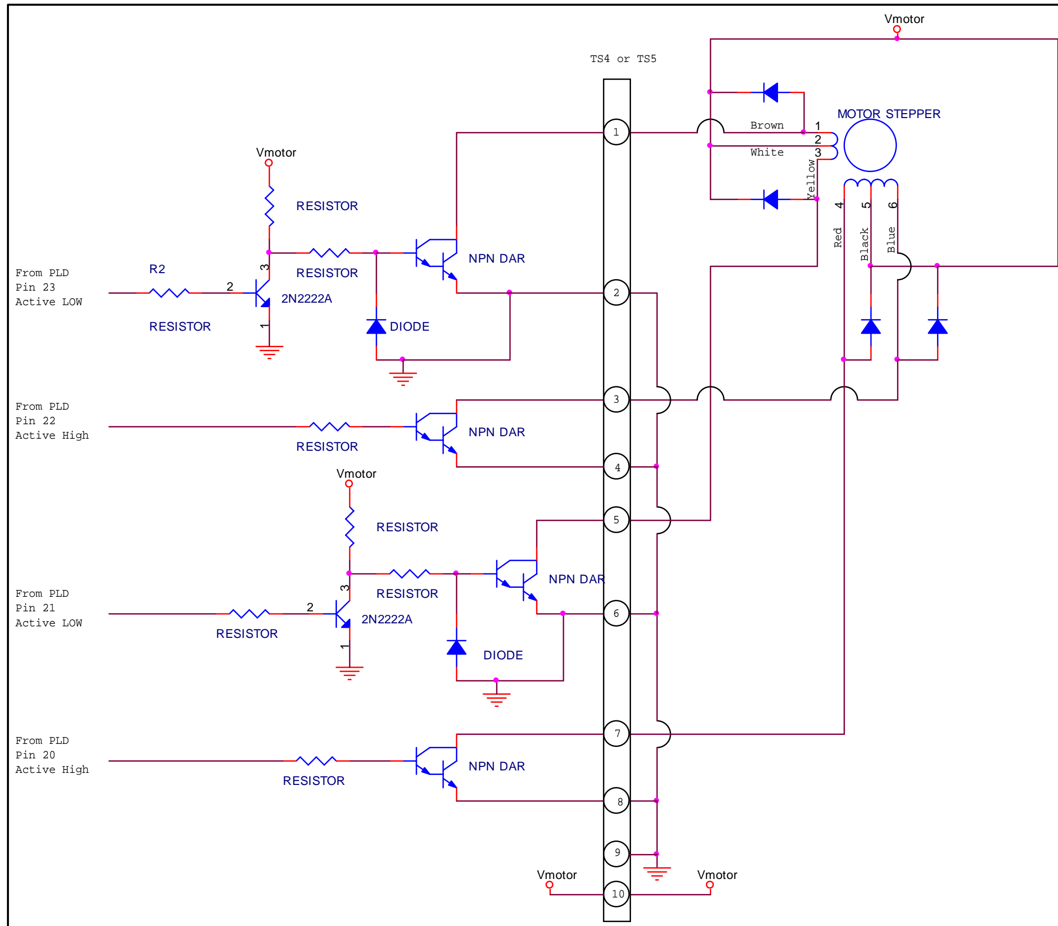


Figure 17. Stepper Motor Configuration

For clarity, the transistor output circuits are shown along with the connections to the terminal strip. The diodes by the stepper motor should be included to prevent back Electro-Motive Force (EMF) from damaging the Darlington pair.

The PAL is programmed so that the motor can work in three different modes of operation. These are selected via 2 lines going into the PAL. See Table IV for bit settings

Table IV. Stepper Motor Mode Select

SS	HT	Stepper Motor Mode of operation
0	0	Half Step
0	1	Not Used (defaults to Half Step)
1	0	Single Step
1	1	High Torque

The SS and HT pins select the step sequence to be executed by the PAL and can be switch at any time via the Motor Control Port. This allows for the motor to use different modes if more positional accuracy or higher torque is needed. Additional information on stepper motors and their modes of operations, see Appendix D.

ii. Software

The stepper motor is controlled by a number of distinct software routines. The first is very similar to the output software already discussed, but in the case of the motor control port, the low and high four bits (also known as nibbles) control Motor 2 and Motor 1 respectively.

For Stepper Motor operation, the nibble is defined as follows:

Table V. Stepper Motor Control Nibble Definitions

BIT		**Description**
MSB	2^{n+3}	SS Bit (See Table IV for more information)
	2^{n+2}	HT BIT (See Table IV for more information)
	2^{n+1}	Coils ON Bit (1 = on, 0 = off)
LSB	2^{n}	Direction (1 = forward, 0 = reverse)

As stated in the hardware section, bits SS and HT select the mode of operation (High Torque, Single Step or Half Step, see Table IV for settings). The Coils On bit gives the programmer the ability to turn off the stepper motor coils. Normally, even when the motor is not turning, at least one coil is on to hold the stepper motor in position. In the majority of applications, this is a desirable trait, but in some instances, like when power consumption is a concern or when the rotor must be able to spin freely, the ability to turn off all the coils is desirable, hence the inclusion of this feature.

If, for example, a %1011 was written to the motor control nibble, the motor's coils would be on, the motor would turn in the forward direction, and it would be in single step mode.

The second routine required generates a clock signal for the stepper motor. This clock determines the speed of the motor rotation. For this the 68HC11's Output Compare (OC) function is be utilized. The OC function uses the Timer Counter Register (TCNT). This register holds a count that is updated at each clock pulse. The OC uses this value and compares it to the value in its own register (TOCx were x is 1 – 5). When properly set up, an interrupt is triggered when TCNT reaches TOCx and runs the interrupt service routine for that OC. In this case the Interrupt Service Routine (ISR) then sets up for the next pulse edge. The flowchart for the ISR is as follows:

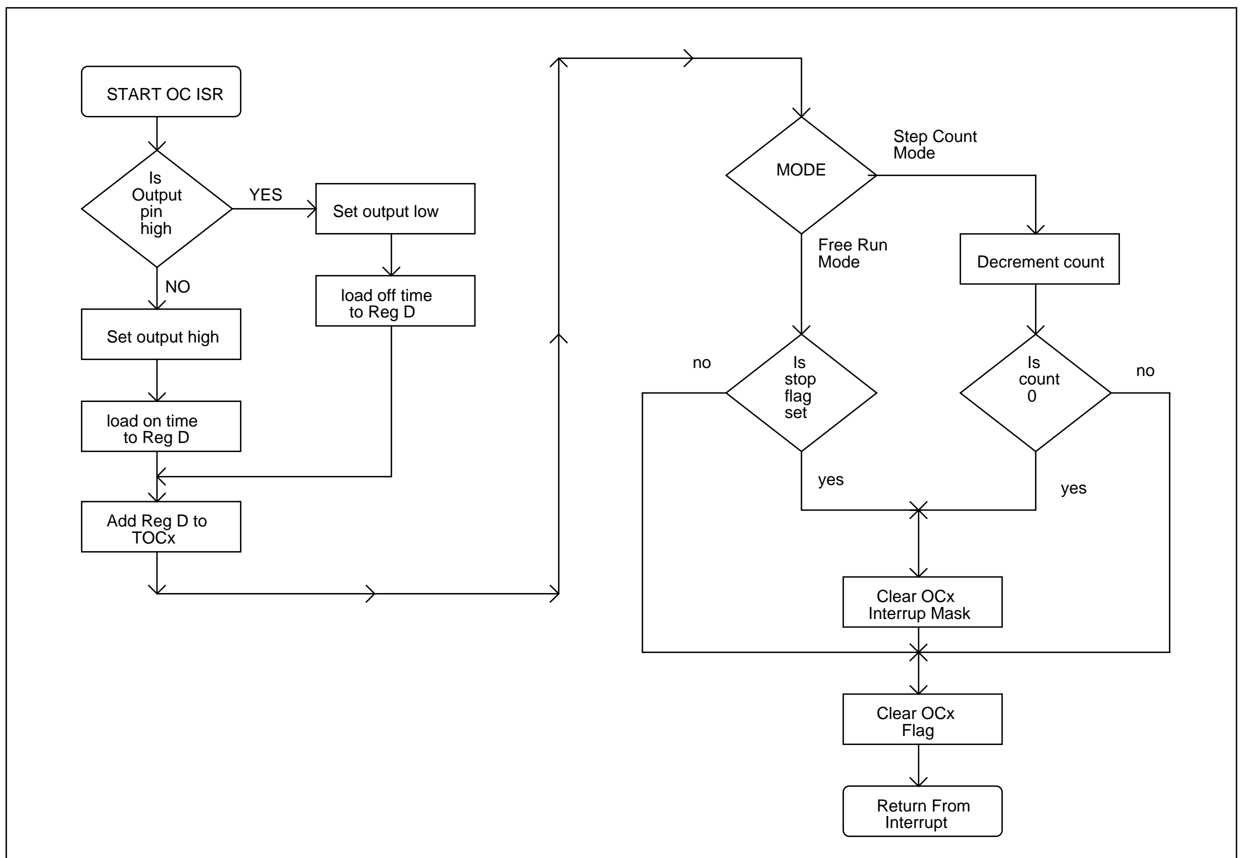


Figure 18. Output Compare ISR

Since the ISR is very time critical, assembly language is preferred for this section of code to minimize any latency caused by code delays. For the full code, see Appendix H.

The on and off time is calculated by a separate routine that does the following computations:

(Equation 2a)

$$On_Time = total_time / 100 * duty_cycle$$

(Equation 2b)

$$Off_Time = total_time / 100 * (100 - duty_cycle)$$

Where: On_Time = number of counts representing on time

Total_time = is 1/frequency (in clock counts)

Duty_cycle = duty cycle of the pulse (for stepper motors this should be 50)

Off_Time = number of counts representing off time

To convert clock counts to time, the prescaler (Bits PR1 and PR0 of TMSK2 register) and the crystal frequency are used. For the CME11E9-EVBU the prescaler is 00 (Prescale Factor of 1) and the crystal frequency is 8 MHz give a time per count of 500ns [3].

(Equation 3)

$$Total_Time = counts * time_per_count$$

Where: Total_Time = Time in seconds

time_per_count = 500ns

Since these calculations are be done outside the ISR, the speed of the motor can be changed at any time by manipulating the $total_time$ value. The code for these calculations can be found in Appendix H in the mtr_clock routine of the Motor.C library.

A section of the ISR is also responsible for stopping the motor allowing the programmer has the ability to set the stepper motor to a free run or step count mode of operation. In free run mode, the stepper motor will continue to run until told to stop via an external flag. In step count mode, the ISR is told how many steps to execute and stops when completed. In both cases the ISR is turned off by clearing it's interrupt mask bit to stop the clock, hence stopping the motor.

iii. Testing

Testing the stepper motor operation for the Expanded I/O Card was accomplished with the use of the Motor.C Small-C library (Appendix I) to create a program for supplying the needed clock and input signals to the motor control port. The TEST.C program (Appendix I) file is broken up into sub programs that are chosen via the PROG variable. In the case of stepper motor testing, sub program #3 was used.

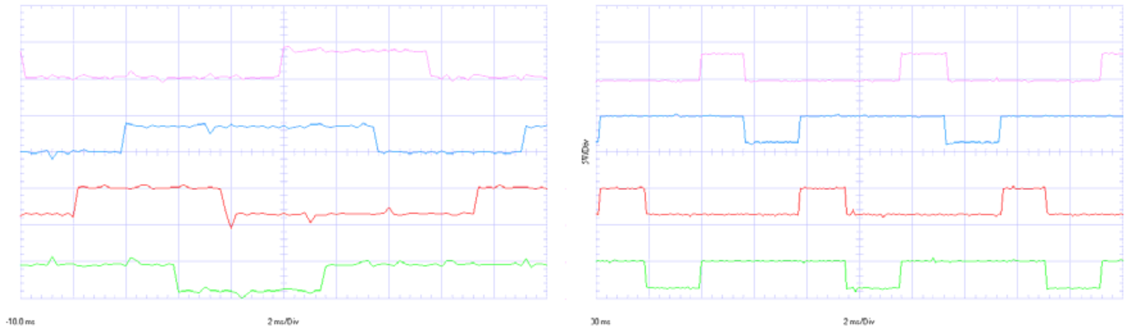
The 3rd sub-program allows the motor to be controlled using the previously tested Input ports. Port 0's lowest two bits set the stepper motor mode, Port 0's 3rd bit sets direction and Port 1 allows the speed to be changed (the higher the value the SLOWER the speed).

After wiring the stepper motor, and verifying the jumpers (as per Figures 17 and 16 respectively) the complied test.c code (TEST.s19) was download and run via Buffalo's GO command (go 2400).

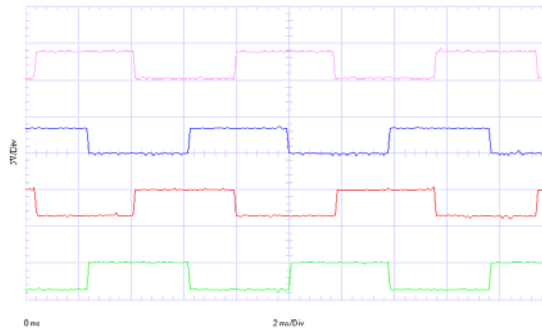
The first test performed was to verify the clock signal on PIN 1 of the PLD associated with the motor in question. See Figure 14 (clock in) for the proper waveform.

Next, verification of the speed and forward reverse was performed using the input test circuits already discussed in section III part C.

The final test was to verify each mode of operation. For this test, a scope was connected to the outputs of the PLD that go to the base of the output transistors or Darlington pairs. Using the first output (pin 23) as a common trigger, the following waveforms were obtained:



(a) Half Step (b) Single Step



(c) High Torque

Figure 19. Sample Stepper Motor Waveforms

Recall that at the PLD output, the 2nd and 4th waves are inverted for proper output from the transistor stage.

To verify the stepper motor clock, a scope was connected to PIN 1 of the PLD (clock in). Then giving the software a known TOTAL TIME and the required 50% duty cycle (as per the software section above) the expected on / off times were calculated and measured using an oscilloscop.

Calculations:

counts = 3315 dec.
time_per_count = 500ns

using equation 3

$$Total_Time = counts * time_per_count$$

substituting into equation 3

$$Total_Time = 3315 counts * 500 ns$$

$$Total_Time = 1.6575 ms$$

Since

Duty_cycle = 50%

Then On_time = Off_time

Using equation 2a / b

$$On_Time = Off_time = Total_time / 100 * duty_cycle$$

substituting into equation 2a / b

$$On_Time = Off_time = 1.6575ms / 100 * 50$$
$$On_Time = Off_time = 0.82875ms$$

Since Small-C only has the can only use integer math, the On_Time and Off_timer above were actually calculated to be 0.825mSec or 1650counts.

When measured on the scope, the following were the results:

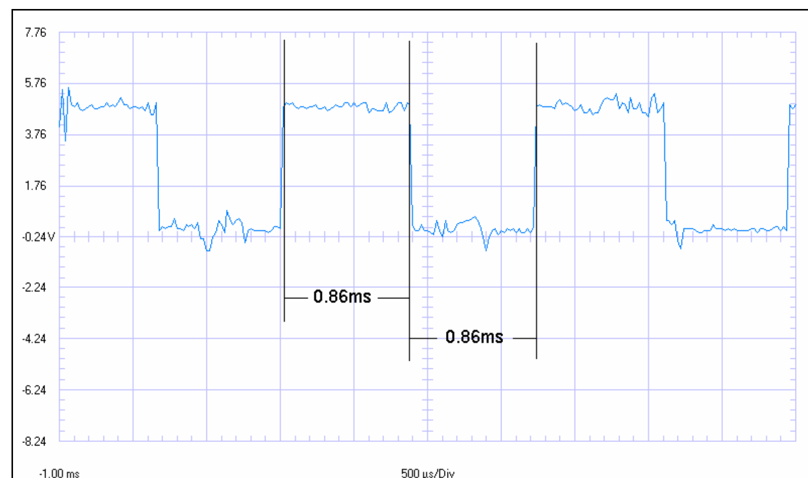


Figure 20. Sample Stepper Motor Clock

The error between the theoretical and the measured values are caused by latency, other software delays and rounding errors associated with integer math. As stated earlier, assembly language routines were used to minimize the latency effect, but software latency cannot be totally eliminated.

It should also be noted that any software that turns off interrupts will interfere with the operation of the clock for the stepper motor effecting the smooth operation of the stepper motor(s).

c. DC Motor Control

i. Hardware

For DC Motor control, jumpers JP2 and / or JP3 should be set according to Figure 16. This configuration sets the following on the PALCE22V10 motor control IC's:

- Pin 2 is connected to the 68HC11 PWM signal (via the Schmitt Trigger Filter) that is generated by the MCU's Output Compare (OC) facility and disables the stepper motor logic.
- Pin 3 of the respective PAL to a HIGH state - selecting DC motor combinational logic.
- Pin 4 is set LOW to indicate motor control (not allowing direct control of the output pins).

DC Motor(s) are connected to the Expanded I/O Card in the following manner:

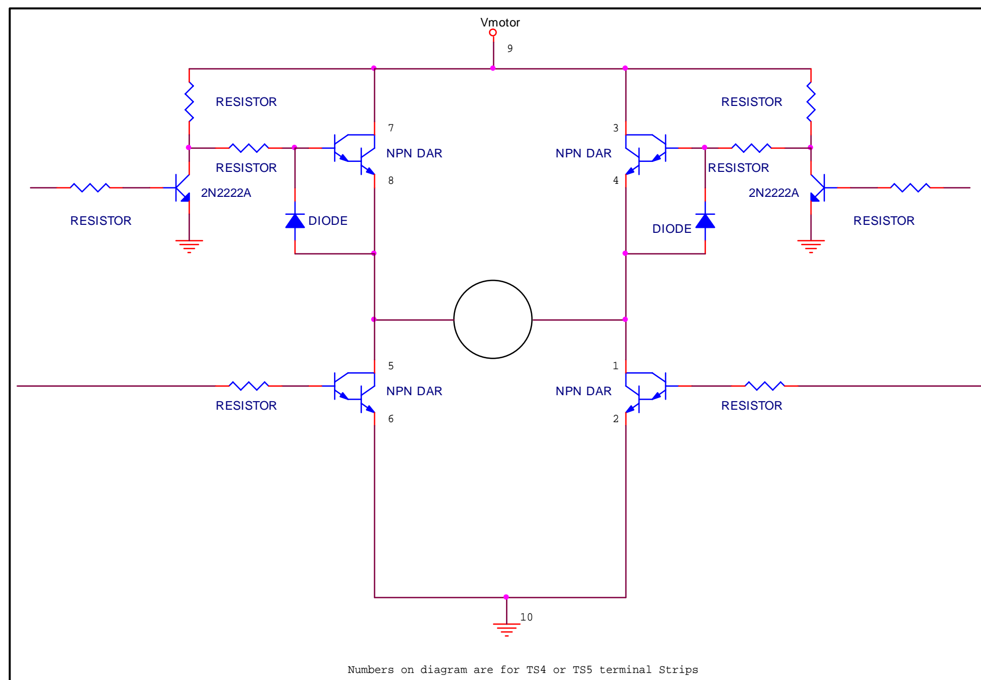


Figure 21. DC Motor Configuration

In the diagram the transistor stage is shown for clarity. The actual connections are made to the terminal strip(s) (TS4 and / or TS5) as indicated via the terminal strip numbers in Figure 20.

ii. Software

DC motors are controlled by the routines for the Stepper Motors discussed above. The same nibble is used to control the DC motor, but in this case the nibble only uses two of the four bits. These bits control on/off and direction only. The control nibble is shown in Table VI.

Table VI. DC Motor Control Nibble Definitions

BIT		Description
MSB	2^{n+3}	Not Used
	2^{n+2}	Not Used
	2^{n+1}	ON Bit (1 = on, 0 = off)
LSB	2^{n}	Direction (1 = forward, 0 = reverse)

For example: writing a %0010 to the nibble would turn the motor on in the reverse direction.

To control speed, the same clock pulse generator software used for the stepper motor is utilized once again, but this time the duty_cycle of the pulse manipulated instead of the frequency (as in the stepper motor control) to control the speed. This type of speed control is known as Pulse Width Modulation (PWM). This turns the motor on and off at different rates, the more on time, the faster it will go, the more off time, the slower it will go. Each motor will have a limit to the minimum duty cycle required for the motor to generate enough torque to rotate.

Unlike the Stepper motor, who's clock has to be turned off to stop the motor, the DC motor is stopped by changing the ON bit (2^{n+1}) to a 0. Since this is the case, there is no need to turn off the PWM signal, therefor the ISR should be in free run mode for DC motor control.

iii. Testing

Testing the DC motor operation was accomplished by using Motor.C Small-C library to create a program for supplying the needed clock and input signals to the motor control port. A sub program in TEST.C was created for this perpose. For the test program set the PROG variable to 4 to run the DC motor test program.

The 4th sub-program allows the motor to be controlled using the previously tested Input ports. Port 0's lowest two bits are used, Port 0's 1st bit sets direction and the 2nd bit turns the motor on/off. Port 1 adjusts the PWM signal (All on, or a value of 255 = 100% and a value of 0, all off, is 0%).

To verify the PWM signal, a scope was connected to PIN 1 of the PLD (clock in). Then giving the software a known TOTAL_TIME and a known DUTY_CYCLE the expected on / off times were calculated and compared to determine the difference.

For this test two different Duty Cycle values will be used. Below is shown the calculations for both a 35% and 75% duty cycle:

counts = 65280 dec.
time_per_count = 500ns

using equation 3

$$Total_Time = counts * time_per_count$$

substituting into equation 3

$$Total_Time = 65280 counts * 500 ns$$

$$Total_Time = 32.64 ms$$

Since

	Duty_Cycle = 35%	**Duty_Cycle = 75%**
Using equation 2a	$On_Time = total_time / 100 * duty_cycle$	$On_Time = total_time / 100 * duty_cycle$
Substituting	$On_Time = 32.64ms / 100 * 35$ $On_Time = 11.424ms$	$On_Time = 32.64ms / 100 * 75$ $On_Time = 24.48ms$
Using equation 2b	$Off_Time = total_time / 100 * (100 - duty_cycle)$	$Off_Time = total_time / 100 * (100 - duty_cycle)$
Substituting	$Off_Time = 32.64ms / 100 * (100 - 35)$ $Off_Time = 21.216ms$	$Off_Time = 32.64ms / 100 * (100 - 75)$ $Off_Time = 8.16ms$

Since Small-C only has the can only use integer math, the actual values calculated by the program were slightly different (for 35% on time was 11.41ms and off time was 21.19ms, for 75% on time was 24.4ms and off time was 8.15ms).

The generated waveforms, as seen on Pin 2 of the PLD, for both 35% duty and 75% duty are as follows:

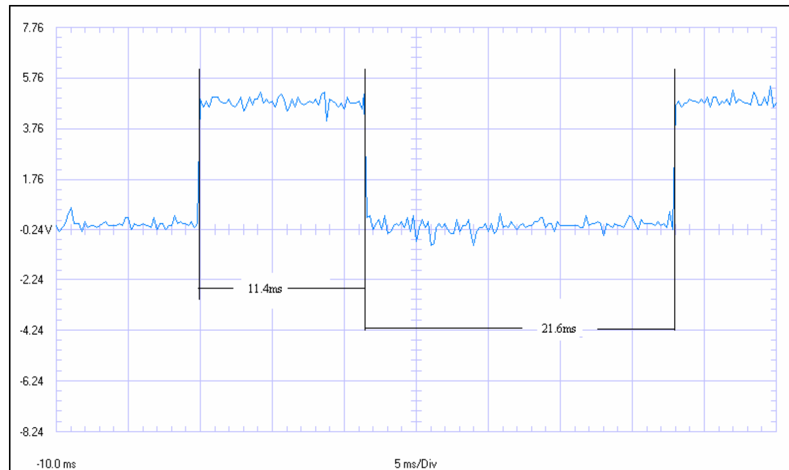


Figure 22. PWM Signal at 35%

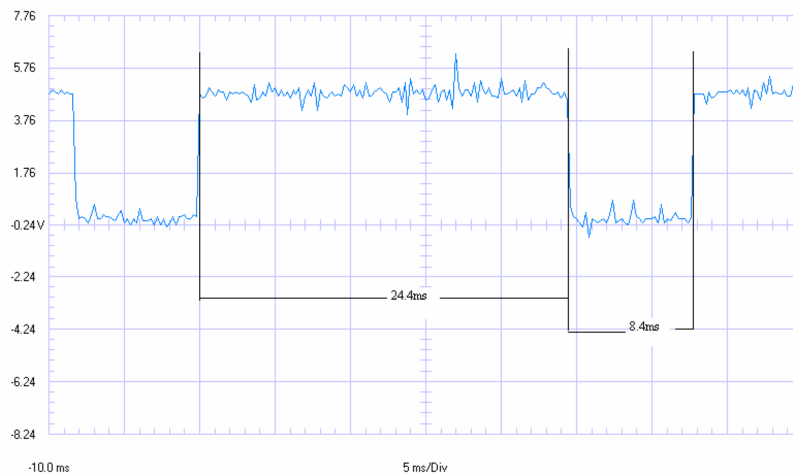


Figure 23. PWM Signal at 75%

The error between the theoretical values and the measured are caused by latency and other software delays as well as the aforementioned integer math errors. As stated earlier, assembly language routines were used to minimize the latency effect, but software latency cannot be totally eliminated.

As with stepper motors, any software that turns off interrupts will interfere with the operation of the clock and effect the PWM signal.

4. Analog Output

a. Modifications from initial design

There were two design changes to the analog output of the Expanded I/O Card since the initial design. The most prevalent is that the IC used was changed from a TLC7225C to a MAX505. The reason for the change was that the TLC7225C was no longer being produced in a DIP package. Luckily the MAX505 was a pin for pin replacement. For full specifications on the MAX505 see Appendix G.

The second change was in the addressing PLD. It was found that the $W\overline{R}$ signal was transitioning from low to high after the A0 and A1 lines had already transitioned. Since the D/A IC needed the addressing lines to remain constant until after the $W\overline{R}$ reached it's high state, the output from the D/A did not always appear on the proper channel. This is shown by the following:

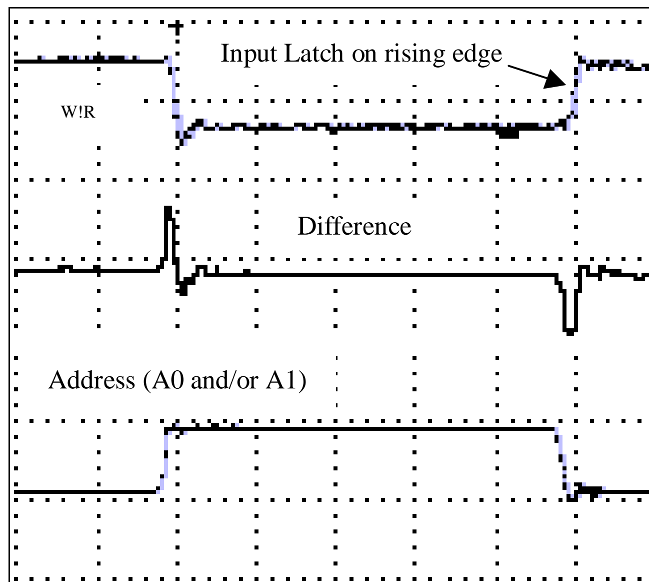


Figure 24. D/A Addressing Timing Error

Two modifications had to be made to the PLD code to correct this problem, the first was to eliminate the propegation delay caused by the use of 2 outputs to create the $W\overline{R}$ signal. The original WinCupl code for the D/A Addressing was as follows:

IO4 = !(!CSn & !RW & A3 & !A2 & !A1 & !A0);

IO5 = !(IO6 # IO7);

IO6 = (!CSn & !RW & !A3 & A2 & A1 & !A0) # (!CSn & !RW & !A3 & A2 & A1 & A0);

IO7 = (!CSn & !RW & !A3 & A2 & !A1 & A0) # (!CSn & !RW & !A3 & A2 & A1 & A0);

Since the IO5 output uses IO6 and IO7, these outputs had to go from the output back to the fuse array and then through extra gates to form the output for IO5, adding propagation delays. The IO5 logic was changed to the following:

IO5 = !(!CSn & !RW & !A3 & A2);

Which is IO5 written out with all the logic used to generate IO6 and IO7 substituted into the equation then minimized. This removes the propagation delay from the $W\overline{R}$ signal.

The second PLD modification was to ADD a delay by putting the old A0 and A1 outputs to spare pins and then connecting the A0 and A1 pins to the new pins, essentially doing the opposite of the code changes done for IO5 and hence adding propegation delays to hold IO6 and IO7 for an extra period of time allowing $W\overline{R}$ to rise before the transition of A0 and A1.

These changes produced the following improved outputs:

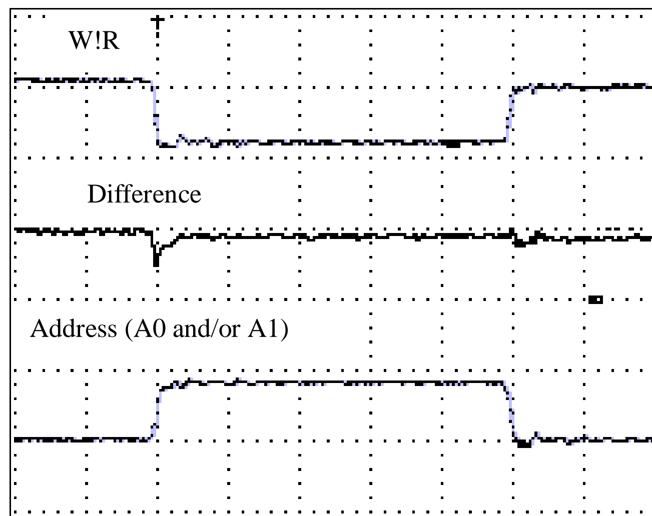


Figure 25. Corrected D to A Addressing

b. Hardware

The analog output circuit is based around the MAX505 Quadruple 8-Bit Digital-to-Analog Converter (Appendix G), as shown in Figure 26.

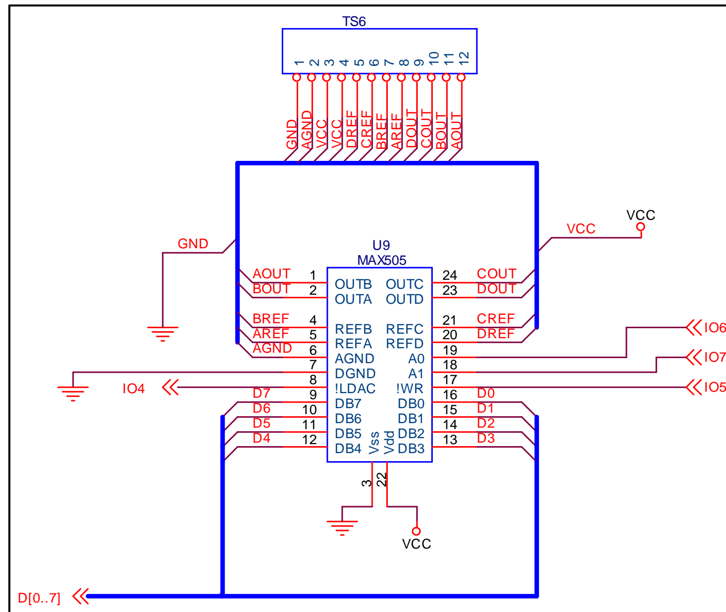


Figure 26. D/A Converter

Since it is a multiplexed device, the IC receives four control inputs from the PAL22V10 used for addressing. For this IC, the PAL really does not supply an address signal, but it supplies a series of control lines. IO6 and IO7 direct the incoming data D[0..7] to the converter for the proper output (OUTA..OUTD). This is shown below:

Table VII. Definition of Outputs from Addressing PAL To D/A Converter

Peripheral Port Address	$W\overline{R}$ (IO5)	A1 (IO7)	A0 (IO6)	Output
$B5n4	1	0	0	OUTA
$B5n5	1	0	1	OUTB
$B5n6	1	1	0	OUTC
$B5n7	1	1	1	OUTD

IO6 (D/A $\overline{WR}$) indicates that data is available on the data bus for the D/A. The last line controlled by IO4 of the Addressing PAL is the $\overline{LDAC}$ line. This line will be sent low while address $B5n8 is being accessed. This line places the new analog values on the output pins of the D/A converter.

c. Testing Analog Output Addressing

Testing the addressing for the D to A converter is more complex. For this test enter the following test code at address 2400:

```
LDAA B5n0
STAA B5n4
STAA B5n5
STAA B5n6
STAA B5n7
JMP 2400
```

The first opcode (LDAA B5n0) allows the IO0 line to be used for scope triggering, allowing signals to be compared via a fixed signal. This signal is placed on Channel A and the other signals are read in turn on channel B. When the signals are combined on a single graph, the results look as follows:

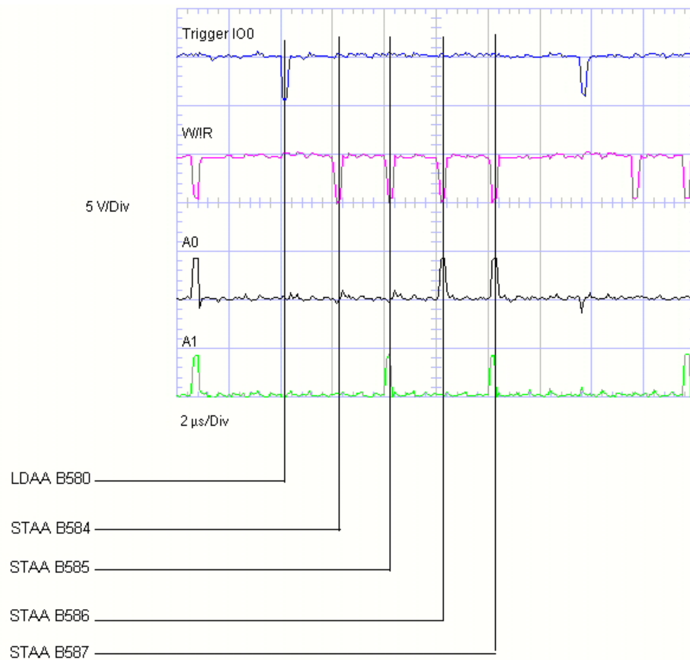


Figure 27. D to A Addressing Sample Waveform

d. Software

To write a value to D/A output pin, the 8 bit representation of the required output must be sent to one of the four addresses shown in Table VII. This can be accomplished using the pokeb()

command in Small C or via a STAA or STAB in assembly language. The formula for the conversion from the 8 bit value to the voltage is as follows:

(Equation 4)

$$Vout = Vref * (x / 256)$$

where: Vout = D/A voltage output

Vref = Reference Voltage

x = value sent

Once the value is sent to the D/A, an additional write must be performed to address $B5n7 to update the output. If this is not done, the output voltage will not change. The value sent is of no significance since this line is derived from addressing only.

e. Testing Analog Output

Each channel of the analog output circuit was tested using the memory modify command in the Buffalo Monitor. Modify each of the four addresses ($B5n4 to $B5n7) by placing a value into the address according to Equation 4. Then modify $B5n8 so that LDAC goes low.

The following chart compares the values sent verses the actual voltage output. The test was conducted with Vref = +5v.

Table VIII. Analog Ouput Test Results

Value Stored (hex)	Value Stored (dec)	Calculated Vout (rounded to 2 dec)	Measured Vout
00	00	0.00V	0.07V
40	64	1.25V	1.23V
80	128	2.50V	2.47V
A0	160	3.13V	3.07V
E0	224	4.38V	4.29V
FF	255	4.98V	4.90V

D. Small C Software Library

To allow for the fastest software development time, three header files and three software library files were created (see Appendix H).

The first of these files is B-VECTOR.H. When included in a Small-C program (using the #include <B-VECTOR.H> statement), names for each of the interrupts can be used in the program in place of interrupt jump vector addresses. This allows for improved readability of code as well as eliminating possible address errors.

Next, the DEFINE.H file allows the use of 68HC11 Port Names instead of addresses. In this file, both Small-C define statements and ASM Equates are used. It was determined that, when embedded ASM code was used in Small-C, the listed ports in Small-C's define statements were not able to be translated in the 2nd stage of compiling. The ASM equate statements solved this problem.

The BUF-SUB.H file defines the entry points of useful Buffalo routines.

The first software library, DEBUG.C has four functions to assist in debugging and data display. By far the most useful function is DEBUG() itself. When placed in the main loop of the code, the routine will look for incoming characters from the RS-232 port and either store and/or transmit information as per the following flowchart:

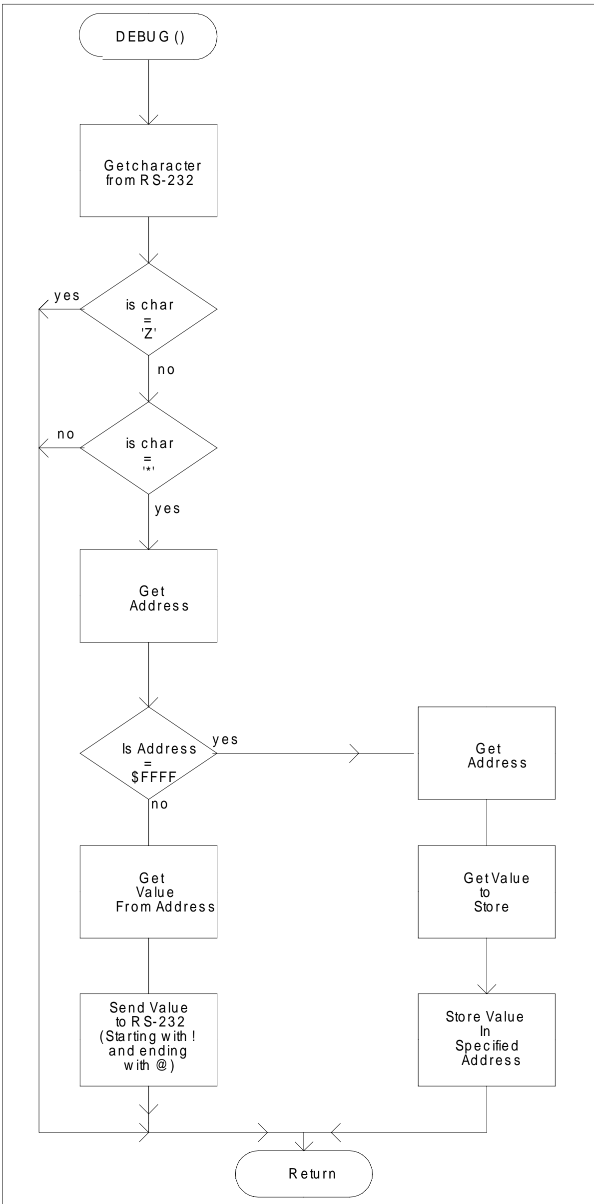


Figure 28. Flowchart For Debug()

The other routines print either a single character, a hex value from a given address, or print a cr/lf to the RS-232 port. For a full listing see Appendix H

The Motor.C Library contains all the functions necessary to drive DC and / or stepper motors. This library has a total of 13 sub-routines. Since this library contains interrupt routines that must be first for the compiler to interpret, this library must be before all other code libraries [except for the Startup.C (supplied with the Small C Compiler) and the '.H' files] at the beginning of Small-C programs. See Test.C in Appendix I for example code.

The routines included in the Motor.C library setup the motor control arrays (A and B) as well as calculate motor variables, turn motors and coils on and off, create the clock pulses, and define the direction of the motor. For a full list of the routines and functions see the Motor.C library in Appendix H .

The last library is for the Analog to Digital and Digital to Analog converters. These routines set up the Expanded I/O Cards D/A and the 68HC11's A/D converts and allow input/output to/from those ports. Refer to Appendix H.

E. Test Software

Sample code has been included in Appendix I to show how to integrate the various libraries listed above. This code was the code used for testing the Motor Control Port as well as other functions and shows the use of all the Library routines.

F. CME-PC Visual Basic Interface Program

This Visual Basic program and was written as an interface for this project, but can be used for any project on a 68HC11 development board running the Buffalo monitor. This program allows for the user to download .S19 program files to the 68HC11. When the download is completed the code will automatically be executed (the software assumes a origin of 2400hex). It also acts as an interactive interface (used with the DEBUG routine listed above) to allow for the display and modification of memory / registers during the execution of 68HC11 programs. For more information on the use of this program, see Appendix J.

References

[1] Motorola Inc. *M68HC11 Reference Manual*, Motorola Inc. 1991 Section 2-6.

[2] Axiom Manufacturing Inc. *CME11E9-EBVU Drawing AX-SCH-0221 Rev B*. August 16, 1999

[3] Motorola Inc. *M68HC11 Reference Manual*, Motorola Inc. 1991 Table 10-1 page 10-9

Appendices

Appendix A	Axiom CME11E9-EVBU Schematic
Appendix B	Expanded I/O Card Schematics, Specifications and Port Wiring
Appendix C	Addressing PLD Documentation
Appendix D	Motor Control Port PLD Documentation
Appendix E	74HC14 Hex Schmitt Trigger Inverter
Appendix F	2N2222A Transistor
Appendix G	MAX505 Quad 8-bit DAC
Appendix H	Small C Software Libraries and Modules
Appendix I	Test.C Code
Appendix J	CME-PC Visual Basic Interface Program
Appendix K	Project Cost Breakdown