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 400

By
Dan Kohn
December 6, 1999

Table of Contents

Table of Figures

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 (Appendix A) consists of a 68HC11 Microcontroller Unit – MCU (Appendix B), 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.

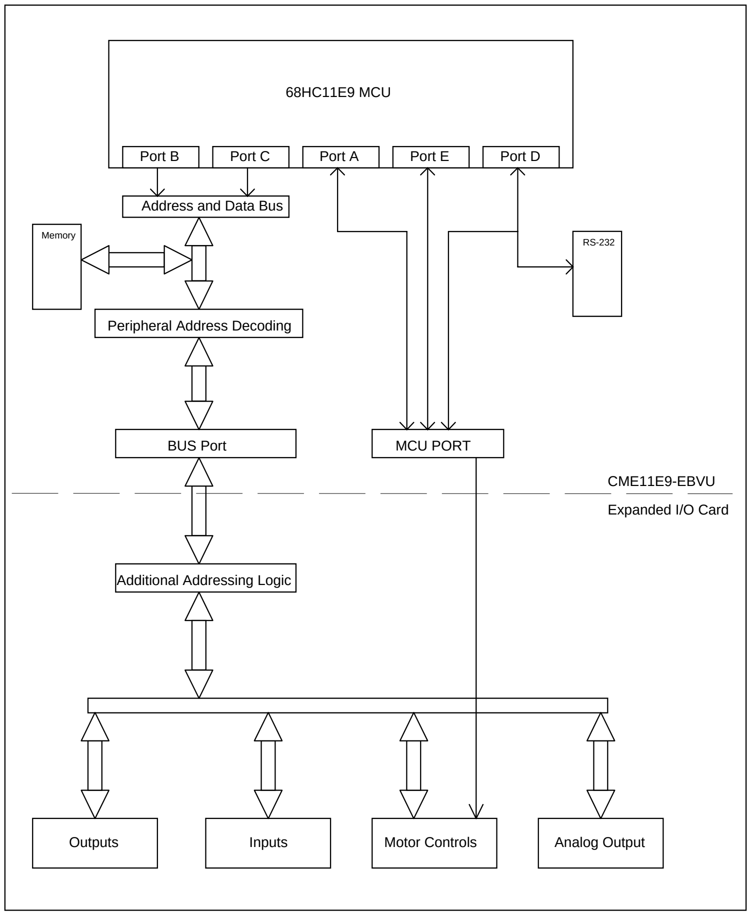

Figure 1. System Block Diagram

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 (Appendix C), 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 D.

B. CME11E9-EVBU Peripheral Addressing and the Expanded I/O Card

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 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 (Appendix E) is programmed with the necessary logic for all the addressing needs of the

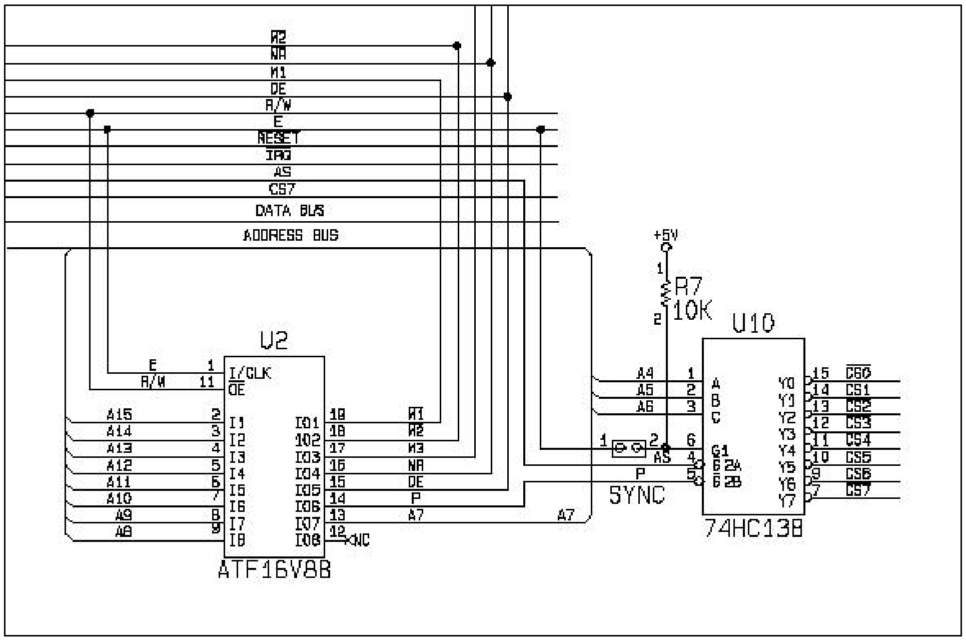

Figure 2. CME11E9-EVBU Peripheral Addressing Circuit [2]

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 \$B580 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 Appendix F) 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.

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 rotary switch is used to select the Peripheral Address line ($\overline{CS0}..\overline{CS7}$) to be used.

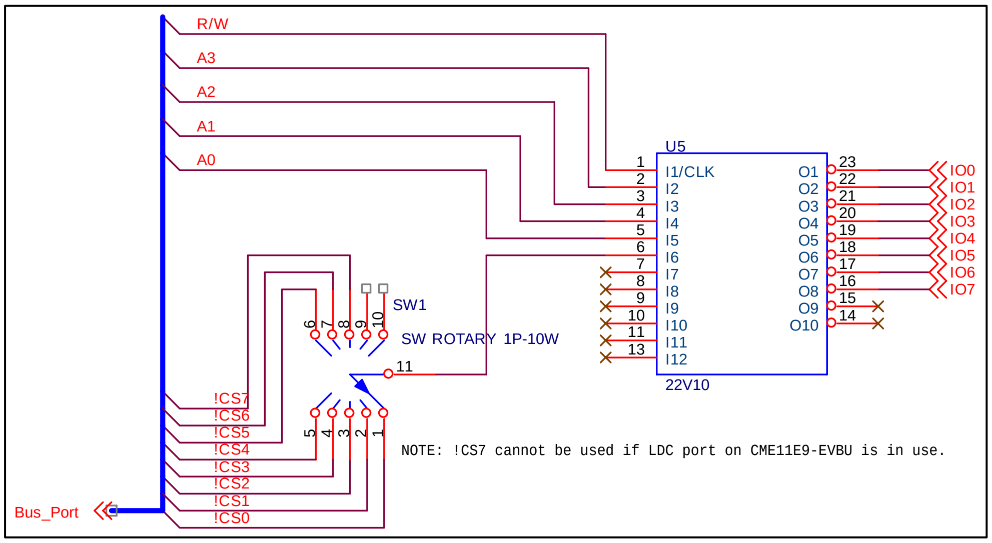

Figure 3. Expanded I/O Card Addressing Decoder

This was done to allow the Expanded I/O Card to be connected to Development Boards that already have some of the peripheral addresses in use. The selected peripheral address line ($\overline{CSn}$) (where n is the CS line selected by the rotary switch) along with A0..A3 and $\overline{RW}$ are then used as inputs to U5 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	$\overline{CSn}$	$R\overline{W}$	A3	A2	A1	A0	Output	Port Type
$B5n0	0	1	0	0	0	0	IO0	Input
$B5n1	0	1	0	0	0	1	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 G.

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

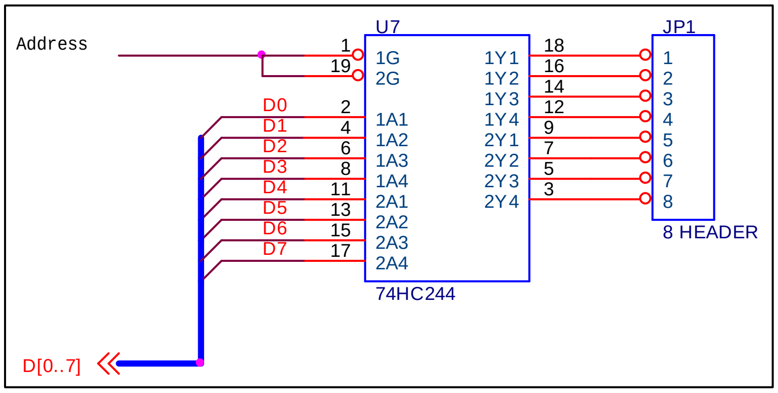

Figure 4. Input Port

buffer / line driver IC (Appendix H) to transfer the data on the input lines (JP1 in Figure 4) onto the data bus when the Address line (IO0 or IO1 in Figure 3) goes low. This makes the data on JP1 directly accessible to the software by the use of memory fetching commands.

b. 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 peek() 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:

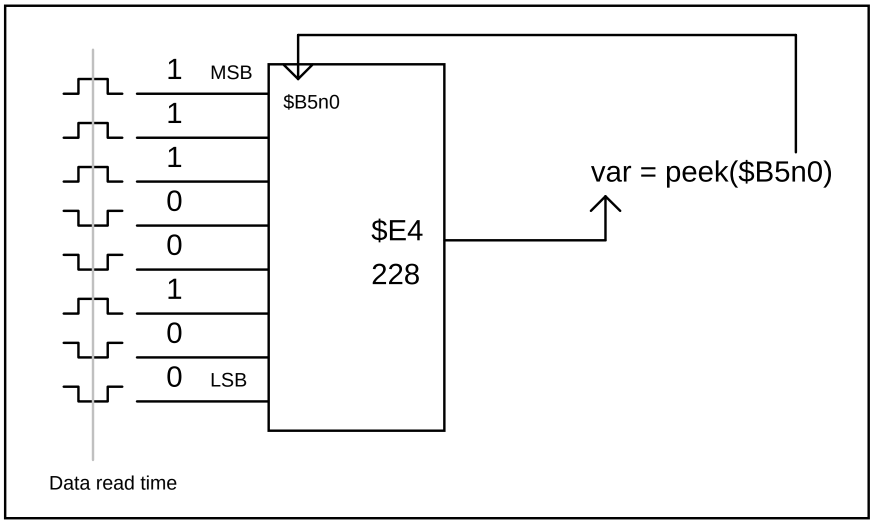

Figure 5. Input Port Example

As seen in Figure 5, 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 poke() or STA, will have no effect on that address.

2. Output Port

a. Hardware

Figure 6 shows the configuration of the output port. The 74HC374 is an Octal D-type flip-flop; positive edge-trigger; 3-state (Appendix I). When triggered via the IO2 line from the PALCE22V10, the information on the data bus will be latched onto the outputs (Q0..Q7).

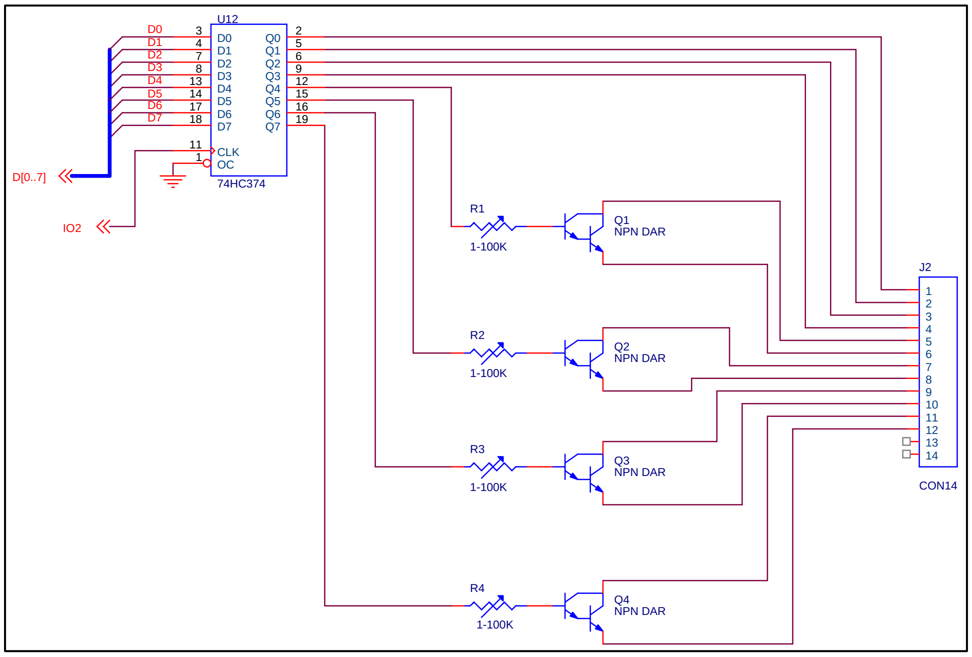

Figure 6. Output Port

To allow for higher voltage and/or current devices to be interfaced to the board, output Q4..Q7 are connected to TIP120s (Appendix J). 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. Since the base current needed for saturation is dependent on the load connected, variable resistors were used to permit adjustment to insure proper operation. The TIP120's collectors and emitters are brought directly to a connector so that any connection configuration can be achieved.

b. Software

Data can be placed on the Output Port of the Expanded I/O Card by use of Small C's poke() function or assembly's STA instruction. In the example in Figure 7,

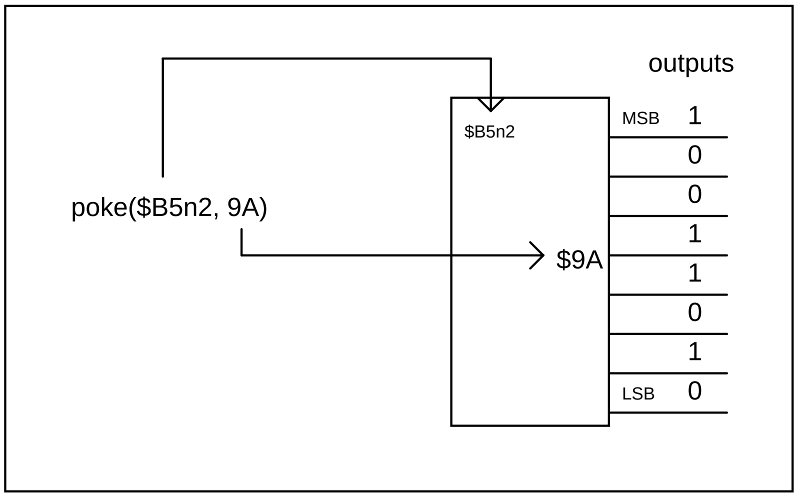

Figure 7. Output Port Example

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 8) uses the 74HC374 for the same application as in the output port discussed above. This provides a latched output to the two PALCE22V10s (Appendix K). 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 L.

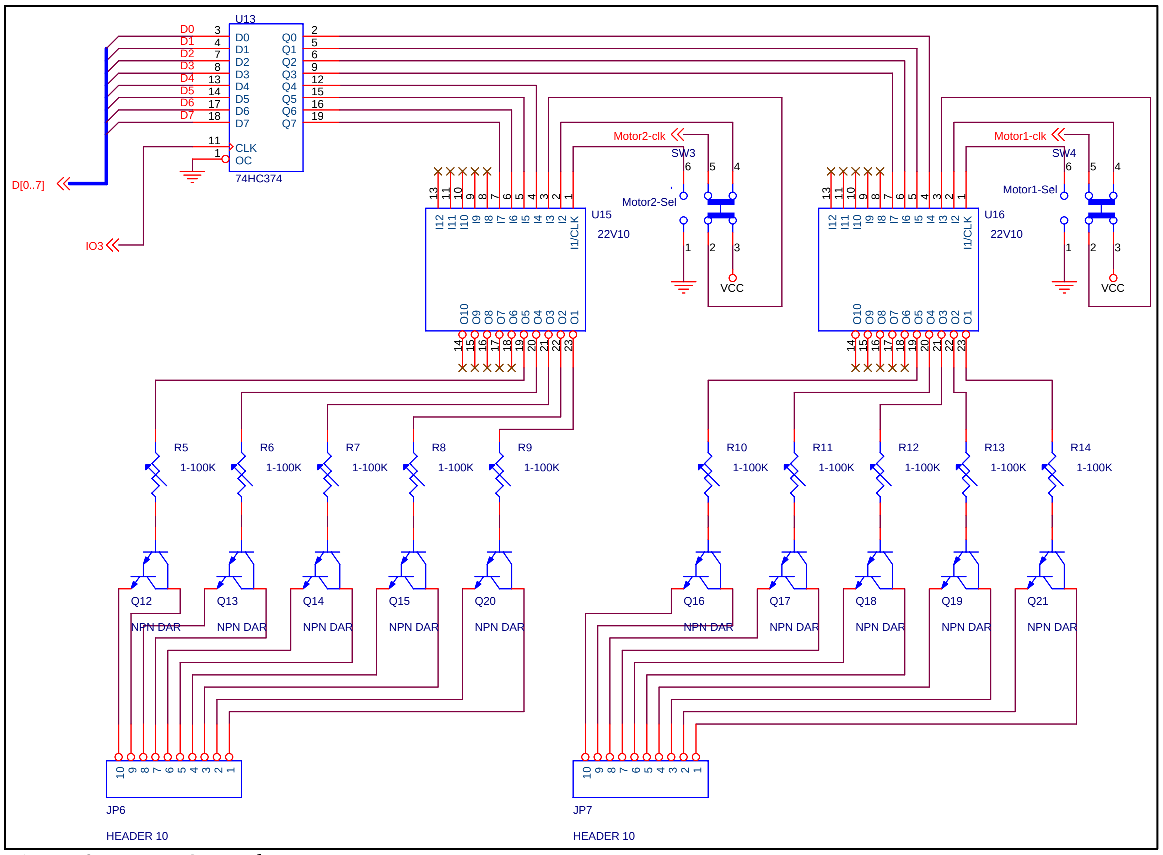

Figure 8. Motor Control Port

a. Stepper Motor Control

i. Hardware

For Unipolar Stepper Motor control, the selector switch (Motor1-Sel and/or Motor2-Sel) is switch to the left (as show in Figure 8). This connects Pin 1 of the PAL to the 68H811 clock pulse (square wave) will be generated by the MCU’s Output Compare (OC) facility and disables the DC motor’s PWM output to/from the PAL. This prevents possible problems that can arise when the motor type and software driver do not match. The switch also sets pin 3 of the respective PAL to a LOW state. This is used by the PAL logic to select stepper motor sequential logic instead of DC motor combinational logic.

The stepper motor(s) will be connected to JP6 and/or JP7 as per Figure 9

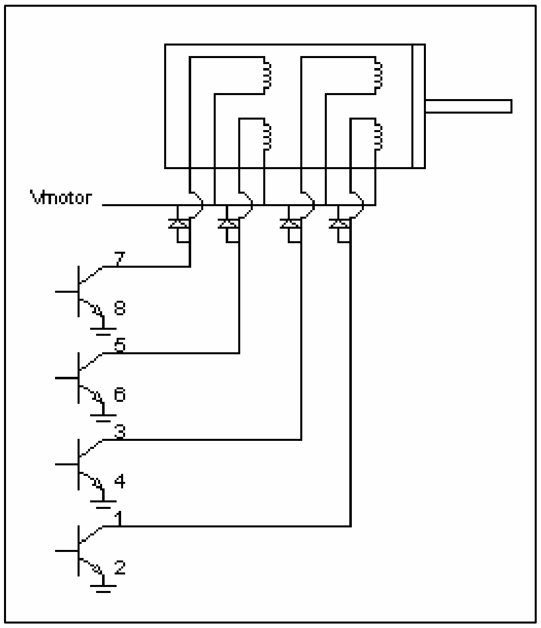

Figure 9. Stepper Motor Configuration

The diagram just shows the last stage of the Darlington Transistor pair (Numbers are from JP6 and/or JP7). External diodes are added to prevent back Electro-motive force (EMF) from damaging the transistors. The diodes chosen should be of sufficient size for the motor chosen. More on this topic can be found in Appendix K.

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 1V. Stepper Motor Mode Select

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

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 M.

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 is written to the motor control nibble, the motor's coils will 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:

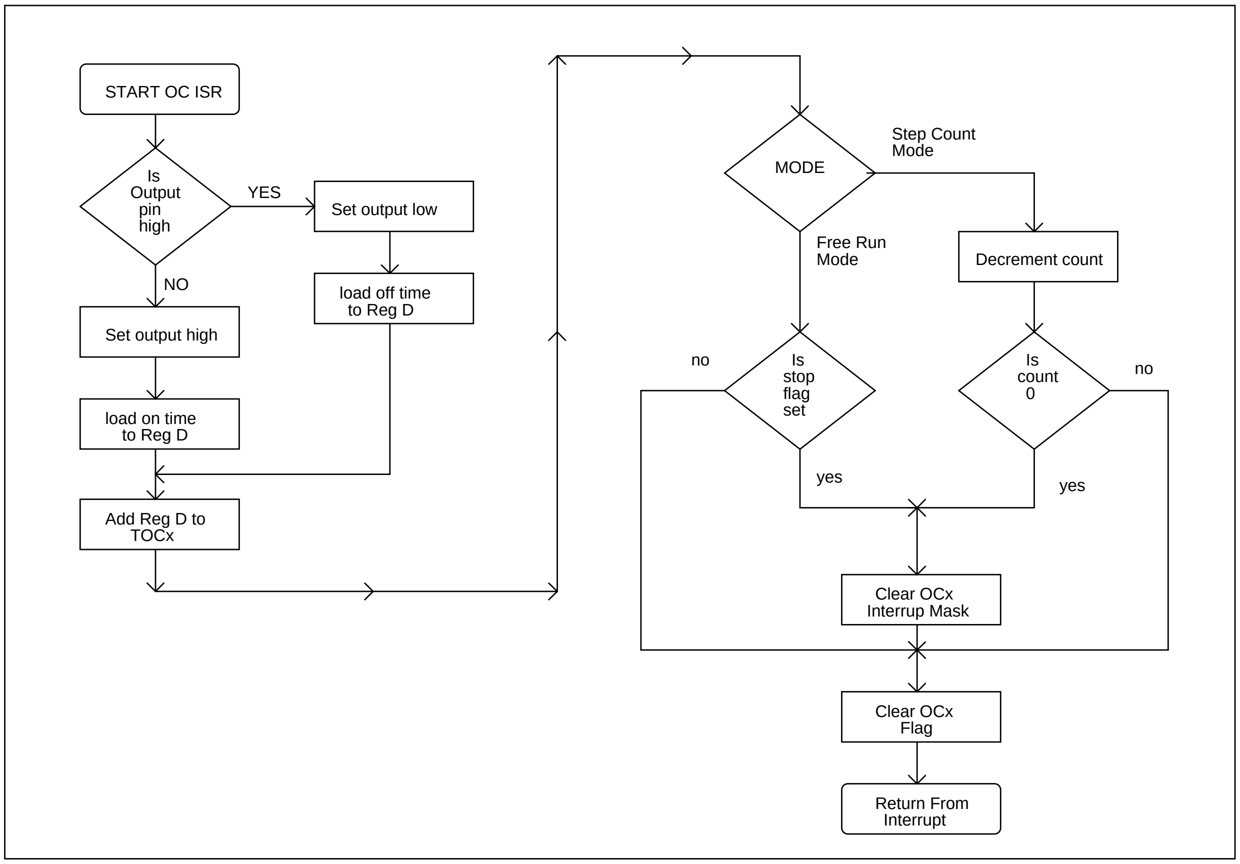

Figure 10. Output Compare ISR

Since the ISR is very time critical, assembly language is preferred for this section of code to minimize any latency caused by code delays. A full description of the OC can be found in Appendix N.

The on and off times are calculated by 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)

$$Time = counts * time\_per\_count$$

Where: Time = Time in seconds

time_per_count = 500ns

Appendix O contains some sample problems using these equations. Since these calculations will be done outside the ISR, the speed of the motor can be changed at any time by manipulating the $total\_time$ value.

A section of the ISR is also responsible for stopping the motor. 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 turns itself off by clearing it's interrupt mask bit to stop the clock, hence stopping the motor.

b. DC Motor Control

i. Hardware

For DC Motor control, the selector switch (Motor1-Sel and/or Motor2-Sel) is switched to the right as show in Figure 8. This connects Pin 2 of the PAL to the 68HC11's Pulse Width Modulated (PWM) signal that is being generated by the MCU's Output Compare (OC) facility. This also disables the clock for the stepper motor to prevent possible problems caused by mismatches between the hardware and software driver. The switch also sets pin 3 of the respective PAL to a HIGH state. This is used by the PAL logic to select the combinational logic for DC Motors instead of the sequential logic for the Stepper motors.

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

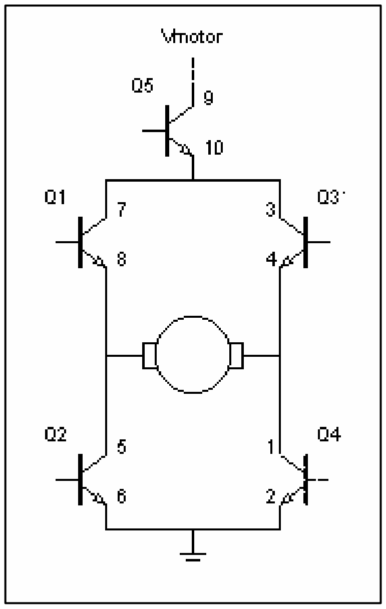

Figure 11. DC Motor Configuration

Once again, the diagram just shows the last stage of the Darlington Transistor pair Numbers are from JP6 and/or JP7. The PAL receives the direction control bit from the 68HC11 and then sends out the proper bit pattern allowing current flow in the proper transistors. Q1 and Q4 allow current flow for the forward direction, Q2 and Q3 for the reverse direction. Q5 is used to control speed.

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 programmer will manipulate the duty_cycle of the pulse instead of the frequency to control the speed. This type of speed control is known as Pulse Width Modulation (PWM). The concept is to turn the motor on and off at different rates, the more on time the motor has, 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. This will have to be found via testing of the individual motor.

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.

4. Analog Output

a. Hardware

The analog output circuit is based around the TLC7225C Quadruple 8-Bit Digital-to-Analog Converter (Appendix P), as shown in Figure 12.

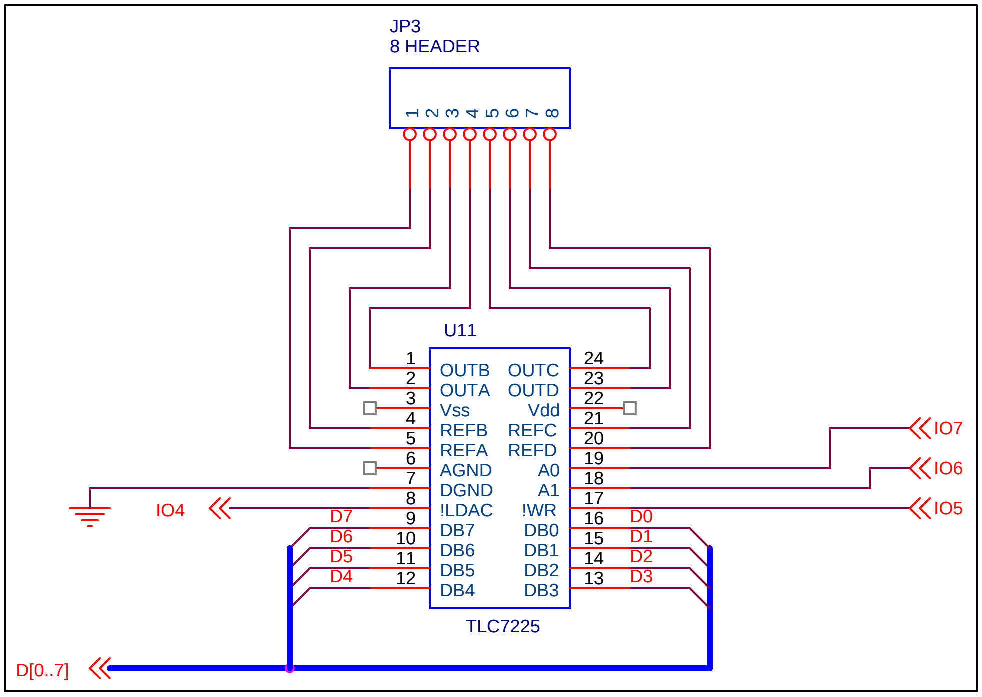

Figure 12. D/A Converter

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)	A0 (IO6)	A1 (IO7)	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.

b. 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 poke() 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.

D. Small C Software Library

All the aforementioned software routines will be part of a Small C library that will be included with the Expanded I/O Card. The library will allow end users to quickly utilize all the features of the Expanded I/O Card by use of modular programming techniques. This will allow for faster development time of software for a wide variety of complex control tasks.

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