Experiment #1
Part A

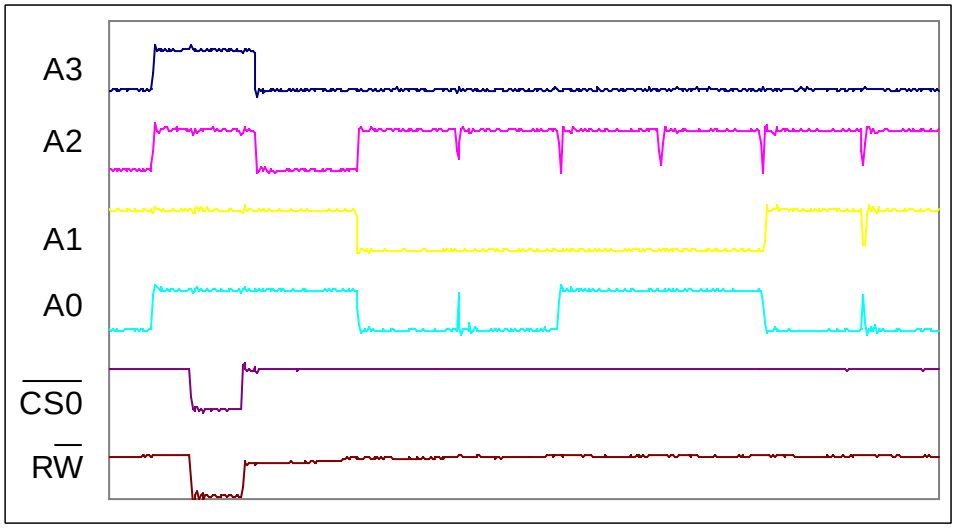
Date Performed: Nov 25, 1999

Purpose: Gain an understanding of the peripheral addressing on the CME11E9-EVBU development board for use in a PAL22V10 .PLD file for programming an address decoder chip.

Procedure: Enter the following program, compile and load to the CME11E9-EVBU development board and then run the program. Hook the channel A of the scope to $\overline{CS0}$ and the other lead to each of the following in turn A3, A2, A1, A0 and $R\overline{W}$ (Trigger off of channel A) and record each waveform.

```
TOP:    STAA    $B58F
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        JMP     TOP
```

Results: The following waveforms were obtained:



Data in file: addr-fluke-1.xls

Conclusions: For the write to address $B58F us the logic equation

$$Adr = NOT(R\overline{W}) \,\&\, NOT(\overline{CS0}) \,\&\, A0 \,\&\, A1 \,\&\, A2 \,\&\, A3$$

Experiment #1
Part B

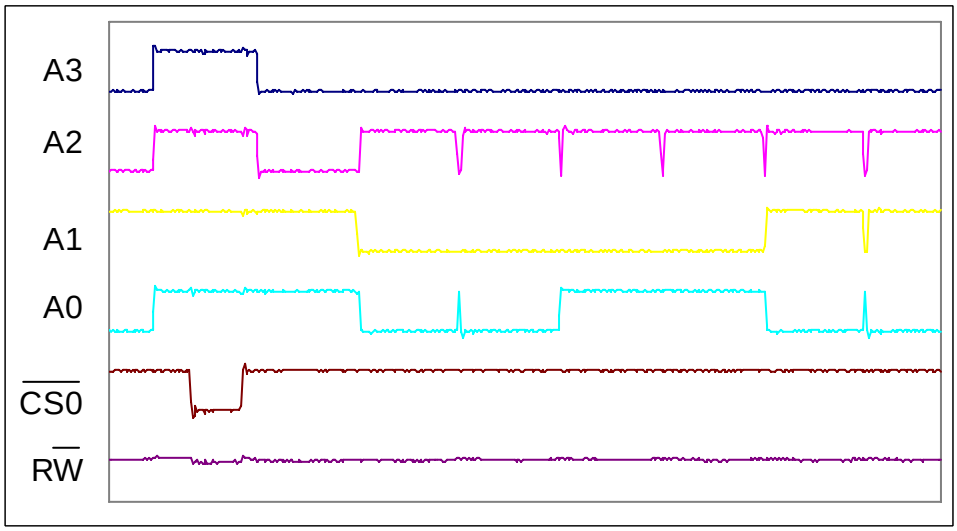
Date Performed: Nov 25, 1999

Purpose: Gain an understanding of the peripheral addressing on the CME11E9-EVBU development board.

Procedure: Enter the following program, compile and load to the CME11E9-EVBU development board and then run the program. Hook the channel A of the scope to $\overline{CS0}$ and the other lead to each of the following in turn A3, A2, A1, A0 and $R\overline{W}$ (Trigger off of channel A) and record each waveform.

```
TOP:    LDAA    $B58F
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        NOP
        JMP     TOP
```

Results: Using an oscilloscope , the following waveforms were obtained:



Data in file: addr-fluke-1.xls

*note: RW was always HIGH

Conclusions: To read from address $B58F us the logic equation

$$Adr = R\overline{W} \,\&\, NOT(\overline{CS0}) \,\&\, A0 \,\&\, A1 \,\&\, A2 \,\&\, A3$$

Experiment #2

Date Performed: Nov 26, 1999

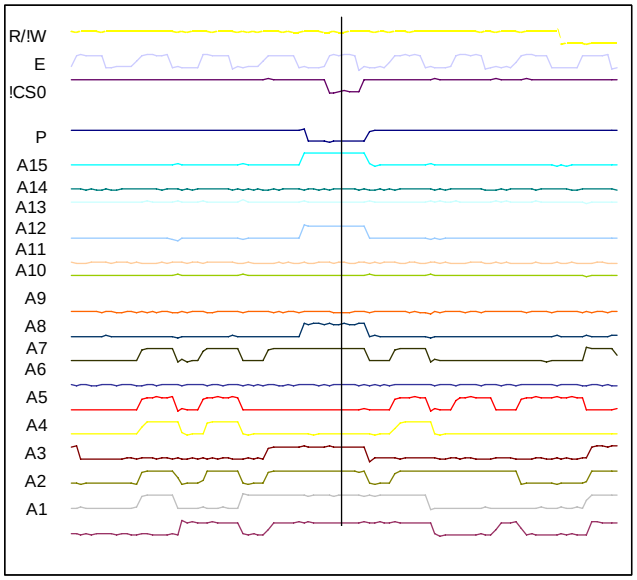
Purpose: Gain an understanding of the Peripheral addressing on the CME11E9-EVBU development board.

Procedure: Enter the following program, compile and load to the CME11E9-EVBU development board and then run the program.

```
TOP:    LDAA    $B58F
        STAA    $2420
        JMP     TOP
```

While the program is running record the following: A0..A15, $\overline{CS0}$, $\overline{RW}$, AS, P, E using $\overline{CS0}$ to trigger.

Results: Using an oscilloscope , the following waveforms were obtained:



Data in file: addr-fluke-2.xls

Conclusions: As show with the vertical line in the graph above, A0..A15 contain the address $B58F during the LDAA $B58F instruction, R/!W is High, P is LOW and E is High. This produces a LOW on CS0.

Addressing for lines A15..A7 is taken care of by ATF16V8B on Axiom Board (with P as the output. P is then used along with A4..A6, AS, and E to produce CS0 using the 74HC138 again on the axiom board.

Experiment #3

Date Performed: Nov 27, 1999

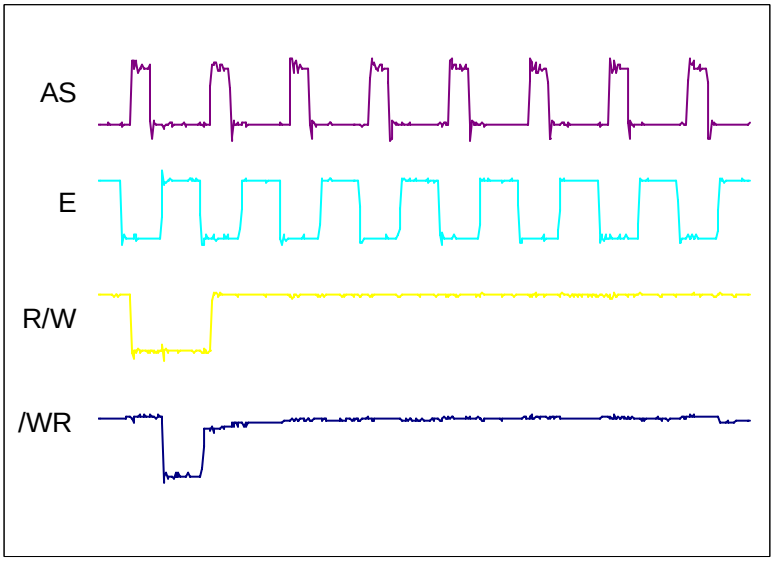
Purpose: Gain an understanding of the Peripheral addressing on the CME11E9-EVBU development board. Especially the difference between pins R/W (Bus_port pin 35) and /WR (Bus_port pin 25).

Procedure: Enter the following program, compile and load to the CME11E9-EVBU development board and then run the program.

```
Top:    LDAA    $B580
        STAA    $B580
        JMP     Top
```

While the program is running record the following: /WR, R/W, E and AS (use $\overline{CS0}$ to trigger).

Results: Using an oscilloscope , the following waveforms were obtained:



Data in file: fluke-3.xls

Conclusions: By referencing the schematic of the Development board, it is noted that the R/W signal is coming directly from the 68HC11, while the /WR is coming from U2. /WR seems to be used as a write enable for the EEPROM and therefore should NOT be used for the addressing of the Expanded I/O Card. R/W should be used for Expanded I/O Card addressing [BUS_PORT PIN 35].

NOTE: Up till this point all tests used the WRONG PIN for the R/W signal (IE they used the /WR pin [Bus_port pin 25]).

Experiment #4

Date Performed: Nov 27, 1999

Background: In the documentation of the CMD11E9-EVBU development board's BUS-Port, CS6 and CS7 are shown without a leading / (like CS0..CS5). This should indicate that these are active high, but other parts of the documentation (mainly the schematic) contradict this.

Also CS7 is also used for the LCD port and it is unclear if the addresses for that port ($B5F0..$B5F1) overlap onto CS7.

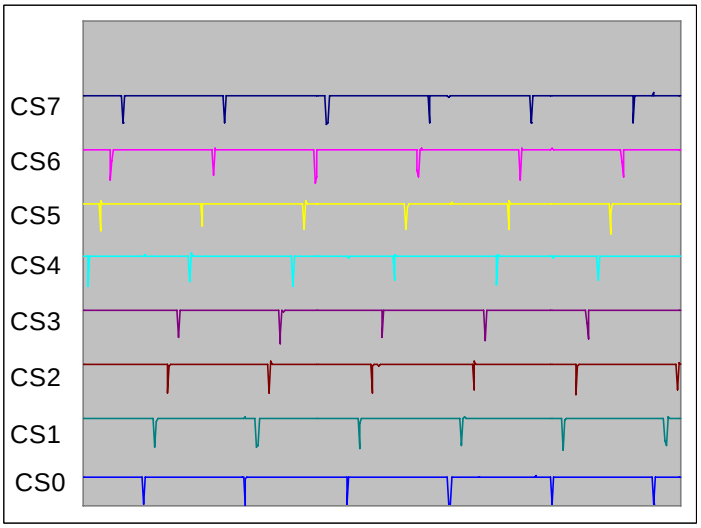
Purpose: Verify that $\overline{CS0}$ .. $\overline{CS7}$ are all ACTIVE LOW and see if addressing the LCD port changes the state of CS7.

Procedure: Enter the following program, compile and load to the CME11E9-EVBU development board and then run the program.

```
Top:    LDAA    $B580
        LDAA    $B590
        LDAA    $B5A0
        LDAA    $B5B0
        LDAA    $B5C0
        LDAA    $B5D0
        LDAA    $B5E0
        LDAA    $B5F0
        JMP     Top
```

While the program is running record the following: $CS0$ .. $CS7$ using $CS0$ to trigger.

Results: Using an oscilloscope , the following waveforms were obtained:



Data in file: fluke-3.xls

Conclusions: This proves that all the CS lines ARE ACTIVE LOW and that address $B5F0 does effect CS7.

Experiment #5

Date Performed: Nov 29, 1999

Purpose: To determine the current draw of a 22V10, and to see if there is a difference in current draw by a programmed vs unprogrammed PAL.

This experiment will also verify the CMEADDR PLD file for IO0..IO4

Procedure: Erase PAL and then hook up power and measure current.

Program PAL with CMEADDR (Shown below)

```
Name     CMEADDR ;
PartNo   00 ;
Date     11/23/1999 ;
Revision 01 ;
Designer Dan Kohn ;
Company   ;
Assembly None ;
Location  ;
Device   p22v10 ;

/* ************************ INPUT PINS ***************************/
PIN   2  = RW                  ; /* R/!W from Bus_Port    */
PIN   3  = CSn                 ; /* CSn from Bus_Port     */
PIN   4  = A0                  ; /* A0 from Bus_Port      */
PIN   5  = A1                  ; /* A1 From Bus_Port      */
PIN   6  = A2                  ; /* A2 from Bus_Port      */
PIN   7  = A3                  ; /* A3 from Bus_Port      */

/* ************************ OUTPUT PINS ***************************/
PIN   23  = IO0                ; /*   See Descriptions    */
PIN   22  = IO1                ; /*       Below           */
PIN   21  = IO2                ;
PIN   20  = IO3                ;
/************* INPUT AND OUTPUT PORT ADDRESSING LOGIC ************/

IO0 = !CSn & RW & !A3 & !A2 & !A1 & !A0       ; /* Input Port 1 */

IO1 = !CSn & RW & !A3 & !A2 & !A1 &  A0       ; /* Input Port 2 */

IO2 = !(!CSn & !RW & !A3 & !A2 & A1 & !A0)    ; /* Output Port  */

IO3 = !(!CSn & !RW & !A3 & !A2 & A1 & A0)     ; /* Motor Port   */
```

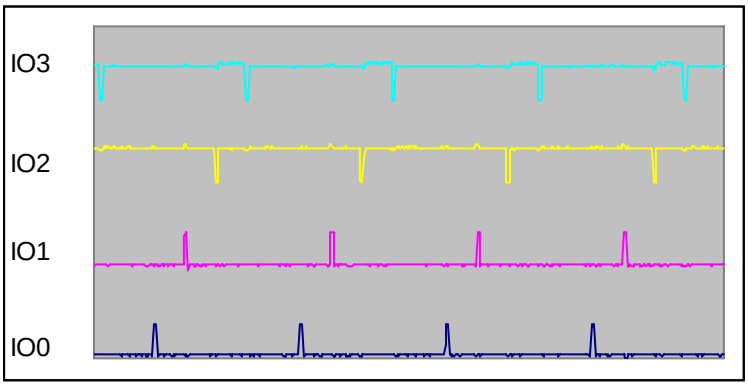
Then connect to CME11E9 board (A0..A3, CS0, RW, V, GND) and program the board as follows:

```
Top:    LDAA    $B580
        LDAA    $B581
        SDAA    $B582
        SDAA    $B583
        JMP     Top
```

Experiment #5 Cont.

Once again read current draw. Also record each output of the PAL. (IO0..IO3)

Results: Measuring the unprogrammed IC the current draw was 31.0mA. The programmed IC drew 48.6mA.



Data in file: fluke-4.xls

Conclusions: Programming the PAL DOES change the current draw of the IC

The CODE for the PAL is correct for the Input ports, output port and motor control port (that is it matches the required signal as stated in the CET400 report)

Experiment #6

Date Performed: Jan 3,2000

Purpose: To test Output circuit (up to but not including the TIP120s).

Procedure: Hook up the Address PAL and a 74HC374 as per the Expanded IO circuit diagram. Connect LED's (Low-Current Red Led's - Radio Shack Cat No. 276-044) to outputs.

1) Using Buffalo's mm (modify Memory) command send various bit patterns to the correct address (B582) and verify the correct bit LED pattern appears.

2) Write a sample C program that modifies the bits.

Results. Found an **error in the circuit diagram**, the addressing IC's input pins start at pin 2 NOT Pin 1.

Once corrected the following were the results:

1) As expected the LED's showed the bit pattern as expected. The only unexpected result is that the Buffalo monitor returns a ROM message (because when it checks to see if the value is updated in memory | see WRITE() routine of Buffalo)

2) Program LED.C does turns on the corner led's one at a time (lines 0,3,4,7) using a TOI ISR to hold the LED's on for a viewable length of time.

Here is the majority of the code:

```
interrupt toi()
{
      sys_clk++;
      pwm_clk++;
      step_clk++;
      bit_set(REG_BASE+TFLG2,0x80);
}

main()
{
      dummy = 1;
      d_int();
      poke (0xD1,toi);
      bit_set(REG_BASE+TFLG2,0x80);
      bit_set(REG_BASE+TMSK2,0x80);
      e_int();
      sys_clk = 0;
      while(dummy = 1)
      {
            if (sys_clk == 10) {pokeb (0xb582,0x1);}
            if (sys_clk == 15) {pokeb (0xb582,0x8);}
            if (sys_clk == 20) {pokeb (0xb582,0x10);}
            if (sys_clk == 25) {pokeb (0xb582,0x80); sys_clk = 0;}
      }
}
```

Program worked well with no unexpected results.

Experiment #7

Date Performed: Jan 9,2000

Purpose: To test Input circuit.

Procedure: Hook up the Address PAL and a 74HC244 as per the Expanded IO circuit diagram. Connect a DIP switch with one side of the switch going to +5v, the other to a 10K resistor going to gnd. Connect the inputs of the 74HC244 to the node between the resistor and DIP switch.

1) Using Buffalo's d (Memory dump) command see the value stored in the address (b580).

2) Add to LED test program, a part that reads the input switches and adjusts the LED flashing accordingly.

Results: 1) Found an **error in the circuit diagram**, the input and output pins are reversed on the circuit diagrams in the report! Refer to Appendix H.

After above problem was corrected the results were still not as expected. Not just the value of B580 change. Instead the range B580..B5FF all changed to match the bit pattern on the Dip switches.

Found an **error in the circuit/PLD programming**. The 244 needs an active high signal, not active low as in the report (again see Appendix H). Will correct the PLD code and try again

Experiment #7b

Date Performed: Jan 10,2000

Results: 1) After fixing the PLD code (CMEADD.PLD Rev. 2.0- added !(…) to IO0 and IO1) the experiment was performed again and worked correctly. This time no unexpected results.

2) I created the program LED-SW.C which contains the following section:

```
temp = peekb(0xb580) ;
 num = temp / 4;
 if (num != old) {sys_clk = 0;}
 old=num;
 if (sys_clk == num) {pokeb (0xb582,0x1);}
 if (sys_clk == num*2) {pokeb (0xb582,0x8);}
 if (sys_clk == num*3) {pokeb (0xb582,0x10);}
 if (sys_clk == num*4) {pokeb (0xb582,0x80); sys_clk = 0;}
```

As expected the rate of led flashing changes when switches are changed.

This proves that the new address chip is functional and that the Input and Output circuits are working and that they do NOT interfere with each other.

Experiment #8

Date Performed: Jan 17,2000

Purpose: To test motor control PAL.

Procedure: Hook up the Addressing PAL, 1 input circuit and one motor control circuit (using LED to indicate output states).

Timing (using the Roll Over Interupt - TOI for timing signal wich is slow enough to be seen). Program Motor-1.C is written so that the input switches can be used to change the state of the control nibble. The DC Motor and Clock inputs have to be moved to change the motor from DC to stepper motor control.

Results: Found a few minor errors in the table showing the bit setup for various stepper motor states. The correct states are as follows:

SS	HT	Mode
0	0	Half Step
0	1	Half Step
1	0	Single Step
1	1	High Torque

All other stepper motor functions worked as predicted.

When testing the DC motor part, it was found that the logic originally used was in error. Each logic statement was outputed to a .d variable (orriginally both the DC and stepper motor both used the clock on pin 1, this was changed because of possilble dammage by Stepper outputs going to an H bridge and possible problems with PWM clocks) But when the conversion was done, the .d's were not dropped. The .d was removed (ver 4.1 of PLD) and the chip re-tested and the chip then worked as predicted in DC mode (stepper mode was also retested with no change in the original results.

Experiment #9

Date Performed: Jan 20,2000

Purpose: Testing of C Software module for DC/Stepper motor control.

Comments: Software worked surprisingly well considering the changes made from previous testing software.

The biggest problem to overcome was the conversion of the array (were the data for each motor is stored) to an output to the motor port. Since an array is stored as words, and each motor uses a nibble for control.

Things Learned: Unlike most C's, small C uses & and | for AND and OR (most C's use && and ||). Small C's compiler does NOT catch these errors during the compile!

Idea for extra Expanded I/O Board Capabilities:

Since I have spare inputs on the PLD chip, by adding a new input, it is possible to make a third mode that would take the DIRECTION, ON, HIGHTORQUE and SINGLESTEP inputs and directly control the 4 transistors. The 5th transistor can be connected directly to the PWM signal input for even more flexibility.

New software library short descriptions:

MOTOR.C	Motor control functions for both DC and stepper motors
DEBUG.C	Debug functions (including memory dump)
B-VECTOR.H	Definitions of Buffalo monitor vector table.
DEFINE.H	C Definitions and ASM Equates for 68HC11 registers.

One Last Note: During this testing it was noted that under certain circumstances (load new code while in stepper motor mode with the coils on but the clock signal NOT active), the PLD's output would change. Investigating, noise (of up to 2v) was found on the clock line. This is probably due to the prototyping (relatively long wires on a digital signal line), but there is no way of telling for sure.

Further investigation is needed!

Experiment #10

Date Performed: Jan 24,2000

Purpose: To investigate "noise" problem found in experiment #9 and find a possible solution.

Step 1: Determine if the "noise" is being caused by the power supply by replacing the power supply with a 12v battery.

Conclusion: Problem still exists with No change in noise.

Step 2: Determine if the noise is being induced by the PLD by placing a filter cap. by the IC.

Conclusion: Problem still exists with NO change in noise.

Step 3: Determine if the noise is induced voltage (60Hz).

Conclusion: Scoped noise, 60Hz detected by relatively small vs rest of noise signal.

Step 4: Tried to shield the clock wire (which is relatively long) by wrapping it with second wire grounded at one end.

Conclusion: Problem still exists with No change in noise.

Step 5: Tried placing a inverter (smitt Trigger - 1/6 74HCT14) in line with signal before PLD.

Conclusion: Change in operation, but still encountering some noise. Problem reduced significantly.

It is possible that the remaining noise is because the signal was inverted (leaving it at a high state when the clock signal is turned off.

Step 6: Put a second inverter in line with the first.

Conclusion: Operation is now as predicted with no unclocked steps encountered.

It is still unclear weather the inverters will have to be added to the final assembly or if the problem can be corrected by IC / Trace placement. Still need to experiment further, but this proves that the initial design is operational.

Experiment #11

Date Performed: Feb 6,2000

Purpose: To test D to A circuit.

Procedure: Hook up the DtoA circuit (including the addressing IC) as show in the CET400 report. Hook VrefA to VrefC to +5v and AGND to ground.

Hook up a scope to the outputs of two channels.

1) Using the Buffalo monitor commands, change the address of the channels that are hooked to the scope and B588 (this address has to be written to so that the DAC Latch line goes low and transfers the latched data to the dac for output).

When B588 is written to, the scope should show the new volatages on the output lines.

2) Enter the following Small C code, compile and load into the CME11E9 board:

```
#code 0x2400
#data 0x2000
#stack 0x1f00

#include <startup.c>
#include <define.h>
#include <debug.c>

unsigned int temp;

main()
{
      for(temp=0;temp<=255;temp++)
      {
            pokeb(0xb584,temp+1);
            pokeb(0xb585,temp+1);
            pokeb(0xb588,1);
            show(0x2001);
      }
showc('*');
crlf();
crlf();
}
```

Put the scope on outputs A and B and set it up for one trace on trigger. Run the program and record the results.

Results:

1) The circuit acted as predicted in the CET400 report. As with previous experiments, the Buffalo monitor returns a ROM message when the addresses are written to, but the data does get transferred out as a new analog value after the B588 address is written to. The output corresponds to the following formula:

$$Vout = (byte\_sent / 255) * Vref$$

2) Using the above program, the results were again as expected. Both outputs A and B were the same, and both showed a ramp as the software counted from 0 to 254. The output remained at near +5v when the program stopped execution. The following graph is the waveform produced: