

**La TROBE UNIVERSITY
DEPARTMENT OF ELECTRONIC ENGINEERING
ELECTRONICS-2**

Microprocessors: Lab 6

**Applications of 68HC11 Computer Hardware
Microprocessor System Design
Stepper Motor Speed and Position Control**

Aims:

Familiarize yourself with the steps in a systems design process.

Explore various instructions and addressing modes of the M68HC11.

Construct and develop a prototype microcomputer stepper-motor control system.

Program & Debug the software to create a working system.

Equipment:

HC-COM, HCCOM INTERFACING BOARD, Stepper Motor Plug Pack power supply, IBM PC,
Software: Stepper.asm, SEQ.asm, Bub.asm

Introduction:

During the electronic systems & design project you studied the torque capabilities of the stepper motor. In this lab we will look at using a stepper motor under microprocessor control for precise positioning.

In a Numerically controlled drilling machine we can use stepper motors to seek from position to position.

To maximise throughput, we want to move the machine drilling platform as little as possible to complete the task in the least possible time. If we moved the machine head in the order that the requests are sequenced it can lead to a large amount of head movement.

In an effort to maximise overall system performance all movement requests can be queued and managed by a device manager that minimises the overall time spent (wasted) moving from position to position. Load and run the program SEQ.S19 to seek to 9 positions in the order they were requested. Record program execution time from commencement until the motor stops rotating.

An improvement in seek time can be gained by sorting all requests into order then position-to-position seek time is minimised. Load and run the program BUB.S19 to seek to the same 9 positions, but in position-sorted order. Again record program execution time from commencement until the motor stops rotating. Is the outcome what you expect? Explain these results.

Finally we will modify assembler software to integrate all these components.

To prevent the stepper motor's current spikes from damaging the 68HC11 micro a separate independent power supply will be used. One supply for the microprocessor and one supply for the motors. To Operate the operational amplifiers we need +/- 12 Volt supplies. There are on-board regulators which require about 3 volts drop across them for proper operation, so Feed +/- 15 volts into the Power connector ST1.

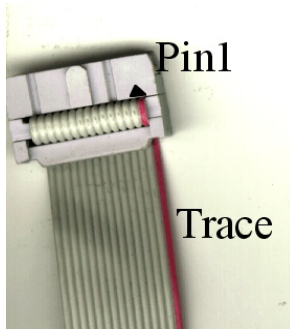
Interfacing Board

Headers:

From the schematic of HC-COM we can see that port A is available via header J2, and Port E via J14. Both headers also provide 5V Power and ground reference, GND.

IDC Header Cables should be connected **straight through**. I.e. Pin 1 to Pin 1, Pin 2 to Pin2, ..., Pin 20 to Pin 20. Ensure that Pin 1 is identified and aligned with the cable trace at both ends.

Ensure that the header connector pin 1 aligns with the header Pin 1 marked on these component overlays. The Wire with the Trace identifies Pin 1.



Also ensure that the headers are aligned exactly as shown in these component overlays. You are dependent upon this alignment to make the connections from the header to the ULN2003A (or MC1413P) using the PCB tracks.

Electrical Analysis

Analyse the OP-Amp circuit below (from Lab 1):

Express V_{ADC0} as a function of the input Voltage on Chan1 and V_{off1} for each SW1 setting.

Assume that $V_{offset} = 0$. Assume the OP Amp and ADC input impedance is Infinite. Also assume that the Voltage drop across DP1 & DP2 diodes is 0.4V. Note: Chan 2 functions identically to Chan1.

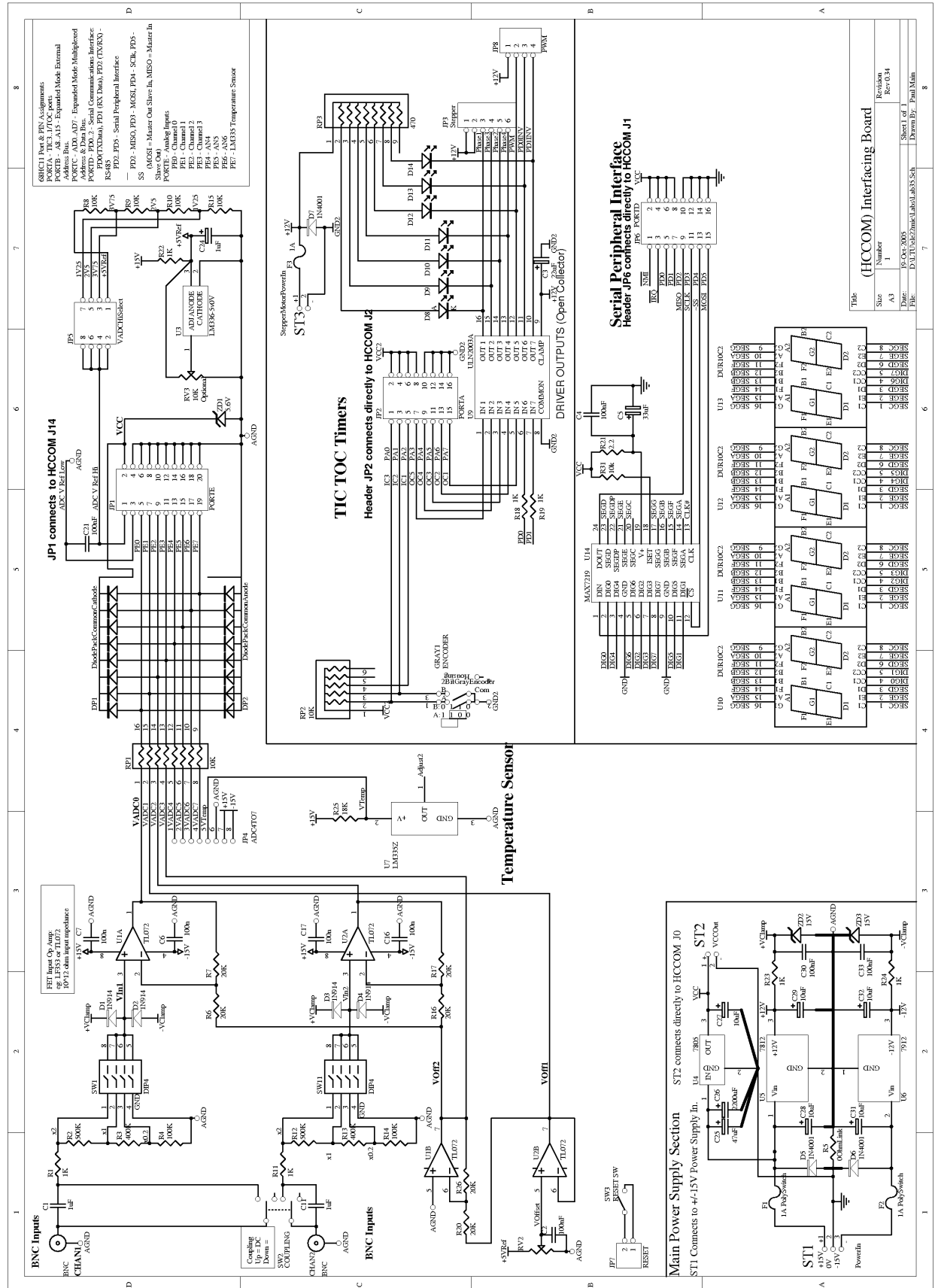
What are the voltage extremes presented at PE0.

Assuming $V_{ADCRefHi} = 5.10V$,

If V_{offset} is adjusted to 2.56 Volts, what will be the input voltage range for PE0 (AN0 input) ?

What will be the value read from the ADC when measuring a Channel 1 input voltage of (assuming the x1 switch is on, $V_{refHi}=5.1V$, $v_{refLo} = 0V$ and $V_{Offset}=2.56V$ and all others are off)

-2.55V, -1.28V, 0V, 1.28V, 2.55V, 5.10V



(HCCOM) Interfacing Board

Title	Number	Revision
Size	1	Rev 0.34
Date	10 Oct 2005	Sheet of 1
File	D:\JUL2005\2meclab\lab35.Sch	Drawn By: Paul Main

PROCEDURES

Ensure the HCCOM PCB to Interface Adapter PCB interconnecting cables are properly connected:

1. Header JP1 to HCCOM J14. (Port E - ADC inputs)
2. Header JP2 to HCCOM J2. (Port A - TIC inputs/TOC outputs)
3. Header JP6 to HCCOM J1. (Port D - PortD - SPI & SCI)

Then connect the stepper motor to header JP3.

SETTING UP THE POWER SUPPLIES

1. Set the bench power supply to **independent** mode.

Connect The interfacing board ST1 to + / - 15V power supply output.

Left = Blue Wire = - 15V,
Centre = Black Wire = GND,
Right = Red Wire = +15V

Connect the 12V plug pack power supply output to **ST3**, and plug in the mains power.

Show a demonstrator your power connections before switching on.

SYSTEM TEST SOFTWARE

2. Load HCCOM with the program STEPPER.S19

Connect HC-COM to a PC

Under buffalo enter: LOAD 2000
 Then in the terminal program, Hyperterminal, send the file stepper.s19
 Send Text File - C:\STEPPER.S19 - F10

After download completes :Execute the code (G 2000)

The displayed delay is the hex value added to TOC count for each step.

TESTING THE STEPPER MOTOR CONTROL:

3. Adjust the potentiometer to the maximum clockwise stepping rate that the stepper can step at without slipping (missing a step).

Using an oscilloscope measure stepping the frequency.

CLOCKWISE ROTATION

4A Measure the step frequency using a CRO _____ Hz

4B What is the Mark-Space ratio for this waveform: _____ %

4C. What is the delay count used for the maximum clockwise rate:

_____ steps/sec

Adjust the potentiometer to the maximum anti-clockwise stepping rate that the stepper can step at without slipping (missing a step).

ANTI-CLOCKWISE ROTATION

4D Measure the step frequency using a CRO _____ Hz

4E What is the delay count used for the maximum clockwise rate: _____ steps/sec

4F Does the motor step equally fast in both directions?

4G Do you think that the stepping rate be increased using Half-Stepping?

5. SEQ.ASM is assembler code to step a fixed number of steps at a set stepping rate to a position. Load and run this code timing how long the code takes. Measure the time taken for the code to seek through the complete sequence. Using a watch with a seconds display, or a stop-watch, Time the program execution from commencement until the motor stops rotating.

6. BUB.ASM is identical to SEQ.ASM except the positions are bubble sorted prior to commencing stepping. Load and run this code, timing how long the code takes.

7. Explain the time difference between the time of 5 and 6.

8. Alter the code **STEPPER.ASM**, from Q2 above, for Half Stepping. Assemble it using the assembler in C:\

Demonstrate your code changes to a demonstrator

8A. What is the maximum stepping frequency now? _____ Hz

8B. What is the delay count used for the maximum clockwise rate: _____ steps/sec

Demonstrate the operating code to a demonstrator

Software Analysis

Refer to the following software - Stepper.asm
(You will need a reference manual to help answer the following questions)

9A. How long after setting the **ADPU bit** must a user wait to use the ADC section and explain why.

9B. What is the function of the **SCAN bit**

9C. What is the function of the **MULT bit**

9D. What is the function of the **CA..CD bits**

9E. How does a **TOC2** Interrupt work?

9F. Identify the Interrupt service routine below, and Highlight the TOC2 Interrupt service routine. How can this code be modified to perform square wave output for assignment 3?

* as11 stepper.asm -1 > stepper.lst

```
*
REGBAS EQU $1000      * Starting address for register block
RAM     EQU $2000
PORTA  EQU $00        * Port A INPUT:PA0..2, OUTPUT:PA3..6, & I/O PA7
*          Port Outputs can be used when not using timer
*          latches specified in TCTL1 (below)
PORTB  EQU $04        * Output port B
PORTD  EQU $08        * Port D data register -, -, SS#, SCK, MOSI, MISO, TxD, RxD
DDRD   EQU $09        * Port D data direction
CFORC  EQU $0B        * Force Output Compare...
OC1M   EQU $0C        * OC1M7, OC1M6, OC1M5, OC1M4; OC1M3, -, -, -
OC1D   EQU $0D        * OC1D7, OC1D6, OC1D5, OC1D4; OC1D3, -, -, -
TOC2   EQU $18        * OC2 register (16 bit)
TCTL1  EQU $20        * OM2, OL2, OM3, OL3; OM4, OL4, OM5, OL5
TMSK1  EQU $22        * OC1I, OC2I, OC3I, OC4I; OC5I, IC1I, IC2I, IC3I
TFLG1  EQU $23        * OC1F, OC2F, OC3F, OC4F; OC5F, IC1F, IC2F, IC3F
TMSK2  EQU $24        * TOI, RTII, PAOVI, PAII; -, -, PR1, PR0
SPCR   EQU $28        * SPI control register
*          "SPIE, SPE, DWOM, MSTR; CPOL, CPHA, SPR1, SPR0"
SPSR   EQU $29        * SPI status register "SPIF, WCOL, -, MODF; -, -, -, -"
SPDR   EQU $2A        * SPI data register On Read-Buffer; On Write-Shifter
```

*** BUFFALO Routine Addresses

```
.OUTA  EQU $FFB8      * Print character in A-reg
.OUTCRL EQU $FFC4     * Output <cr><lf>
.OUTSTO EQU $FFCA     * Output Msg seg (no <cr, lf>)
.OUTSTR EQU $FFC7     * Output Msg w/leading <cr, lf>
OutChar equ $FFAF
```

* HCCOM/BUFFALO Pseudo Vector Equates

```
PVOC2 EQU $00DC      * EVB Pseudo Vector for OC2
```

```
REGS   EQU $1000      * Register stack
TOF    EQU %10000000  * Timer overflow flag
N1     EQU 30         * times for one sec
OPTION EQU $39
ADCTL  EQU $30
ADR1   EQU $31
ADR2   EQU $32
ADR3   EQU $33
ADR4   EQU $34

CCF    EQU %10000000  * Conversion complete flag
ADPU   EQU %10000000  * A/D power up bit
* * * * * Monitor Equates
OUTLHF EQU $FFB2     Print left half
OUTRHF EQU $FFB5     Print right half
CRLF   EQU $FFC4     Print CRLF
```

*** RAM Variable Assignments

```
ORG RAM      Start variables in EVB RAM (upper half)
BRA START
HDLY RMB 2   Half-cycle delay (in 0.5mS increments)
ADCVAL RMB 1
StepDirn RMB 2
```

```
StepPtr RMB 2
* Stepper motor phase activation sequence:
StepTable FCB %0001000, %0100000, %0010000, %1000000
StepEnd   FCB 1
```

```
ADRESULTS RMB 4
```

```
ADCPrompt fcc 'ADC1..4:'
```

```

        fcb 0
DelayPrompt fcc ', Delay:'
        fcb 0

```

START:

```

TOP5   LDS   #$0047          * Top of User's Stack area on HCCOM

        LDD   #$0
        STD   StepDirn      * Step Direction = 0 => stop stepping

        LDD   #$FFF0
        STD   HDLY

        LDAA  #$7E          * Jump (extended) Opcode
        STAA  PVOC2         * Pseudo Vector see manual text
        LDX   #SV5OC2       * Address of OC2 service routine
        STX   PVOC2+1       * Finish jump instruc to TOC2 svc

        LDX   #REGBAS       * Point to register block
        LDAA  #%00000000     *
        STAA  OC1M,X        * Output Compare Mask = 0 for all OCs
        LDAA  #%00000000     * OM2:OL2 = 0:1
        STAA  TCTL1,X       * SET All timers not connected to outputs

* TFLG1 - Timer Interrupt Flag 1 Register $1023
* BIT NUMBER: 7 6 5 4 3 2 1 0
* FUNCTION:   OC1F OC2F OC3F OC4F OC5F IC1F IC2F IC3F
        LDAA  #01000000     * OM2:OL2 = 0:1
        STAA  TFLG1,X       * Clear any pending OC2F Interrupt

* TMSK1 - Timer Interrupt Mask 1 Register $1022
* BIT NUMBER: 7 6 5 4 3 2 1 0
* FUNCTION:   OC1I OC2I OC3I OC4I OC5I IC1I IC2I IC3I
        STAA  TMSK1,X       * Enable OC2 interrupts

        LDX   #StepTable
        STX   StepPtr       * StepPtr -> Start of stepper table values

* ISR Initialisations complete so enable interrupts
        CLI                  * Enable Interrupts

        LDX   StepPtr
        LDAA  0,X
        LDAA  #$FF          * For debugging set to all ones
        LDX   #REGBAS
        STAA  PORTA,X       * Store initial Step position

        LDAA  #11111000
        STAA  OC1D,X
        STAA  CFORC,X

        BSET  OPTION,x ADPU  * Enable Charge Pump Power for ADC
        JSR   delay100us    * Wait for power to stabilise

```

FOREVER:

* Beginning of LOOP-FOREVER

```
        ldx   #REGBAS
```

* Multiple-Channel ADC Operation

* There are two variations in multiple-channel operation.

*

* In the first variation (SCAN = 0), the selected group of four channels are converted,

```

* one time each, results stored in register ADR1 .. ADR4.
* After the fourth conversion is complete, all conversion activity is halted until a
new
* conversion command is written to the ADCTL register.
*
* In the second variation (SCAN = 1), conversions continue to be performed on the
selected
* group of channels with the fifth conversion being stored in register ADR1
* (replacing the earlier conversion result for the first channel in the group),
* the sixth conversion overwrites ADR2, and so on.
*
* A/D Control/Status Register ADCTL = 1030
* All bits in this register may be read or written, except bit 7 which is a read-only
status
* indicator and bit 6 which always reads as a zero.
* BIT Number      Bit 7  6  5   4   3  2  1  0
* Bit Function    CCF 0 SCAN MULT CD CC CB CA ADCTL
* Status on RESET  0  0 U   U   U  U  U  U
* ADC Channel Selects: CA..CD
* These four bits are used to select one of 16 A/D channels (see Table 7-1).
* In multi-channel mode (MULT = 1), the two least-significant channel select bits are
ignored,
* and the CD and CC bits specify which group of four channels are to be converted.
* Channel Result in ADRx
* CD CC (MULT=1)
* 0  0 AN0->ADR1, AN1 -> ADR2, AN2 -> ADR3, AN3 -> ADR4 (Chan 1, Voff1, Chan2, Voff2)
* 0  1 AN4->ADR1, AN5 -> ADR2, AN6 -> ADR3, AN7 -> ADR4 (VADC4..VADC7 from JP4)
* 1  0 Unspecified Results - Reserved.
* 1  1 VRH Pin->ADR1, VRL Pin -> ADR2, (VRH)/2 -> ADR3, Unspecified -> ADR4
* Start ADC the conversion SCAN=0, MULT=1, for AN0..AN3.

```

```

        LDAA    #00010000 * Select conversion of ADC inputs AN0-AN3, results ->
ADR1-ADR4
        STAA    ADCTL,x   * Signal ADC to start conversion

```

```

* Now wait until conversion done
convwait:
        BRCLR   ADCTL,x CCF convwait

```

```

        LDY    #ADR1+REGBAS
        LDX    #ADRESULTS
moveRes
        LDAA   0,y
        STAA   0,x
        inx
        iny
        CMPx   #ADRESULTS+3
        BLO   moveRes

```

```

* Work out stepper motor direction
        LDAA   ADRESULTS * Get ADC converted input voltage value
        SUBA   #$80
        STAA   ADCVAL * Store for later use
        BEQ    IsZero
        BPL    PlusOne
        BMI    MinusOne
IsZero:
        LDD    #0
        BRA    StorDirn * If 0, store 0
PlusOne:
        LDD    #1
        BRA    StorDirn
MinusOne:
        NEGA                   * A = -A
        STAA   ADCVAL * Store for ISR use

```

```

        LDD      #-1
StorDirn:
        STD      StepDirn

* Work out stepper motor speed
        LDAA     ADCVAL      * -128..127
* Take Absolute Value of signed number
        CMPA     #0
        BGE     DONTNEG
        NEGA
DONTNEG

* DELAY MSB = 2 * (7F - ABS(ARDCVAL))
        SBA     #$7F
        ASLA
        NEGA

* Least sig byte limits min delay to 200 counts...
        LDAB     #200

* Save that as the stepping rate
        STD      HDLY      * Speed

* Display a prompt
        ldx     #ADCPrompt
        bsr     outStringX

* Get the 4 ADC results and print them using the BUFFALO Monitor
        ldy     #ADRESULTS
allFour
        ldaa    0,y
        bsr     outHexA
        iny
        cmpy   #ADRESULTS+3
        blo    allFour

        ldx     #DelayPrompt
        bsr     outStringX
        ldd     HDLY
        bsr     outHexD

        JSR     CRLF      * output carriage return - line feed

        bSR     delay100us

        BRA     FOREVER * loop forever

***
* SV50C2 - Output Compare 2 service routine
*
* Called at each OC2 interrupt.
***
SV50C2 LDD HDLY          * Get delay time for 1/2 cycle
        LDX #REGBAS
        ADDD TOC2,X      * Add HDLY to last compare value
        STD TOC2,X      * Update OC2 (schedule next edge)
        BCLR TFLG1,X $BF * Clear OC2F

        LDD StepPtr      * X = Table Pointer
        ADDD StepDirn    * 1, 0 or -1
        XGDX

        CPX #StepEnd     * If at end of table, then wrap to start
        BLT NoWrap

```

```

        LDX #StepTable      * by Re-Loading the starting address of table
        BRA StorePtr
NoWrap:
        CPX #StepTable      * If before start of table, then wrap back to end
        BGE NoWrapBack
        LDX #StepEnd-1     * by Loading the ending address of table
NoWrapBack:
StorePtr:
        STX StepPtr        * Save next address
        LDAA 0,X           * A = Next Stepper Value

        LDX #REGBAS
        STAA REGBAS+PORTA    * activate next stepper coil

        RTI                ** Return from OC2 service **

```

```

* -----
* Console Output - an ASCIIZ string pointed to by Index register X.
* -----

```

outStringX:

```

        psha
        pshb
        pshx
again:
        ldaa 0,x           * get the character pointed to by X
        cmpa #0            * Compare with null - character 0
        beq foundNull     * found null character so break out of loop
        jsr OutChar       * output character in Acc. A
        inx
        bra again

foundNull:
        pulx
        pulb
        pula
        rts

```

```

* -----
* Output 2 hex digit version of Accumulator A
* -----

```

outHexA:

```

        psha
        pshb
        pshx
        pshy

        TAB
        JSR OUTLHF * output high nybble
        TBA      * restore it to A
        JSR OUTFHF * output low nybble

        puly
        pulx
        pulb
        pula
        rts

```

```

* -----
* Output 4 hex digit version of Accumulator D
* -----
outHexD:
    psha
    pshb

    pshb
    jsr    outHexA
    pula
    jsr    outHexA

    JSR    CRLF    * output carriage return - line feed

    pulb
    pula
    rts

* -----
* This subroutine delays (a little over) 100 micro-seconds
* -----
* Busy-wait a number of clock cycles
* -----
delay100us:
    psha                * save Accumulator A
*                      * Generate a "short" delay > 100 microsec
    LDAA    #40         * 40 loops for 200 clock cycles at 0.5 micro seconds/clock
DELAYLOOP:
    DECA                *
    BNE     DELAYLOOP

    pula                * restore Accumulator A
    RTS                 *

```

Bonus Questions

Gray Code decoding.

10. Load, debug and demonstrate the Gray code decoder from your assignment 2.

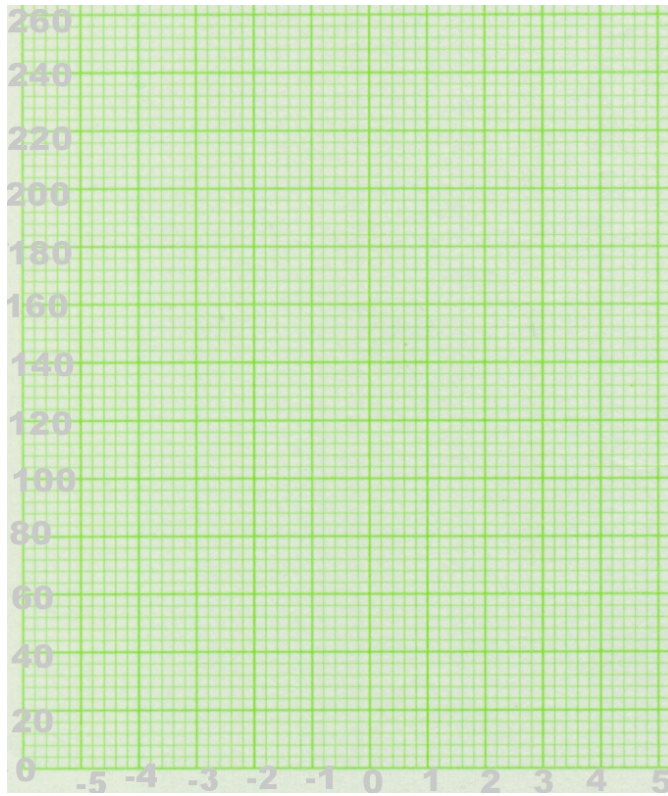
Pulse-Width Modulation.

11. Load, debug and demonstrate the pulse-width modulation code from your assignment 2.

Temperature Measurement & Display.

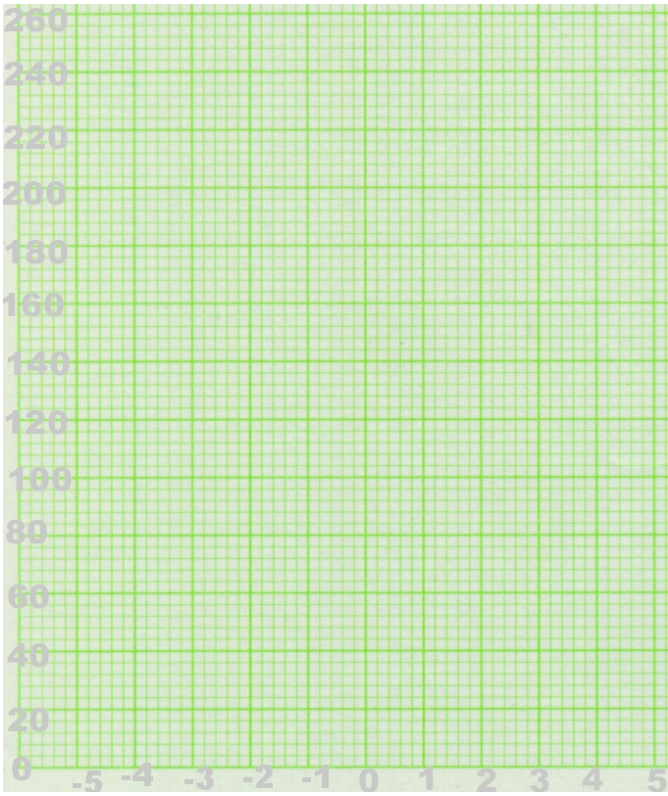
12. Demonstrate the Temperature Measurement and display code from your assignment 2.

ADC
Value:



Analog Input Voltage - VIn1 (VOff1 =0V)

ADC
Value:



Analog Input Voltage - V CHAN1 - (VOff1 =2.56V)