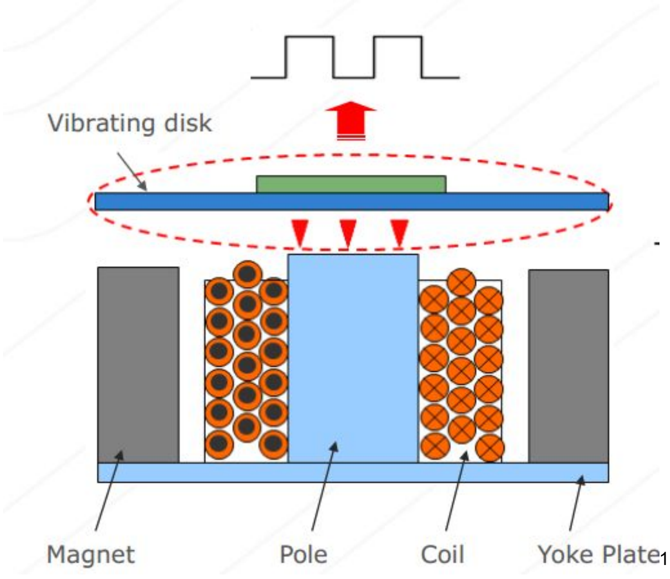
Hardware Theory:

The breadboard will retain no hardware from previous labs except jtag connector. A set of 5 pushbutton, a buzzer and lcd connector will be wired in this week. There are two types of commonly used buzzers, piezo electric and magnetic buzzers. A piezo electric buzzer works by applying a voltage to a iron based plate, which will enable it to expand and contract rapidly producing sound. A magnetic buzzer works by vibrating a plate up and down using magnetic field.



[1]

LCD display is used in this lab. An advantage over the 7segment display is that, LCD display enables to display more and all characters, thus any text or a 48 character long sentence can be written. EA DOG 163 LCD display is being used. The main advantage of using this lcd display is that, the backlight and display are separate. This will enable the user to adjust the backlight without affecting the display. There are 6 different adjustments for the contrast. The backlight led is controlled by blc in the spi bus. The LCD is extremely compact at 55 x 31 mm at the standard font size of 5.57 mm (2 x 16)[2]. The thin size helps reduce the space and weight for the appliances using such displays. The lcd display is

[1] image credits: http://www.cui.com/product/resource/buzzers-buzzersounds.pdf
[2] credits - http://www.lcd-module.com/eng/pdf/doma/dog-me.pdf

connected to the chip using a serial peripheral interface (SPI) bus. The usual spi bus only consists of MISO, SCK, SS, MOSI, but the SPI bus for this lcd consists of an additional RS (register select) and BLC. The RS is used to identify between data and commands, in the signal sent. The SPI connected attaching the chip is considered the slave and one connecting to the output or lcd display is referred as master. MISO (Master input slave output) is not used in this lab, indicating microprocessor could only write on the lcd, but not read from it. The MOSI(master output slave input) is used for sending data to display on the lcd. The SS (slave select) is active low. SCK is shorthand for serial clock. The lcd can display 3 lines with 16 characters long. The lcd only recognizes ascii, requiring the hexadecimal stored in the registers to be modified in order to display the corresponding character. The LCD can be operated either at 3.3v or 5v.[3] For lab purposes, the lcd is operated at 5v, since 5v is easier to output in a digital signal.

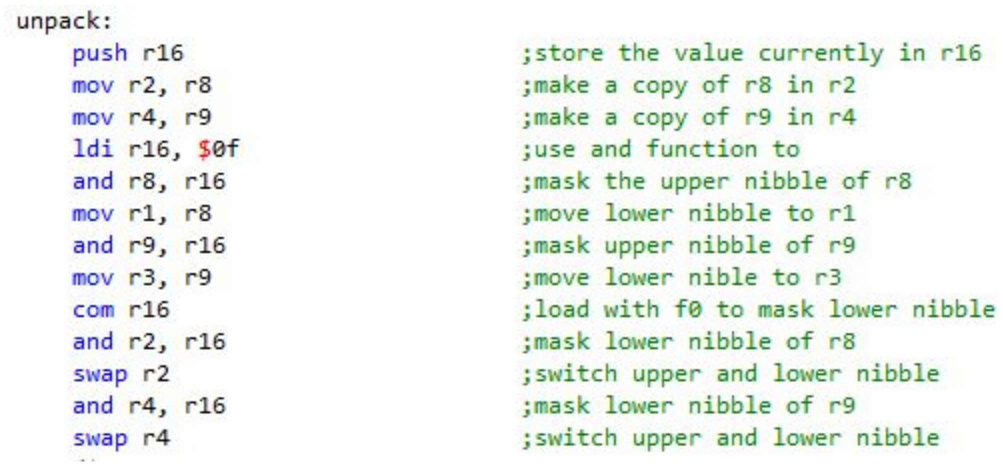
Code Theory:

[3] Credits : reference to EA DOGM163 datasheet

```
freq_meas_1secgate:
        ldi r19, $7B                        ;set initial values
        ldi r18, $13                        ;for the outer loop counter
        in r25, pinA                        ;and positive edge counter
        pos_edge:
                ldi r17, 9                  ;set tweak delay to 10
                in r16, pina                ;take in the current wave signal logic
                cp r16,r25                  ;and compare to previous logic recorded,
                breq neg_edge               ;if it is the same then skip to tweak delay
                dec r17                     ;set tweak delaay for 7
                ldi r25, $00                ;set previous logic to 0 just in case
                brcs neg_edge               ;if there is a carry then branch
                dec r17                     ;if not then decrement
                dec r17                     ;tweak counter twice
                nop
                ldi r25, $80                ;and set previous logic to be 1
                inc r8                      ;and then increment the counter
                brne neg_edge               ;if it didnt over count then go to tweak delay
                inc r9                      ;if so then increment the second register
                neg_edge:
                        dec r18             ;decrement outer loop
                        brne tweak          ;counter and if 0
                        dec r17
                        dec r19             ;then decrement the second register
                        brne tweak          ;if second register is not 0 then keep looping
        ret
                tweak:
                        dec r17             ;decrement the tweak counter
                        brne tweak          ;and keep looping
                rjmp pos_edge
```

The frequency 1 second measurement delay was determined using the logic comparison of the present logic and the previous recorded logic every 32 microseconds. When the comparison between the two registers indicate that they are equal then the loop counter is decremented and the tweak delay is implemented. Since the operation is a comparison between the previous and the present, if the comparison shows that they are equal then it means that the logic has not changed and there was no positive edge detected. The positive edge is detected when the comparison between r16 and r25 does not yield a carry, which indicates that r16 is greater than r25 meaning a logic 1 was detected where it was previously a logic 0, in which the positive edge counter r9:r8 is incremented. In each branch, the tweak delay is adjusted through decrements. Each time there is a change in wave input, then the current value will be recorded into r25. Since each decrement is a decrease of 3 clock cycles, then nops will be added in adjustment for the total clock cycles per loop. In total, there should be 32 clock cycles every loop. The tweak delay adjusts the number of clock cycles depending on where the code branches. For every two clock cycles, a decrement is added into the branch. When r19 reaches 0 then, the code will have completed 1 second of analysis. Every time a 0 is detected for inc and dec operations on r8 and r18 respectively, it goes on to increment/decrement r9/r19. This 0 is detected with the brne operation.

The frequency read from the above subroutine is unpacked, and each nibble of r8 and r9 is assigned to r1 - r4 registers. This is done so each registers can be loaded as a character in the lcd. The unpack subroutine will copy the values of r8 and r9 to two more registers and masked upper nibbles and lower nibbles using the and function. Values in upper nibble are swapped to get it in the lower nibble.

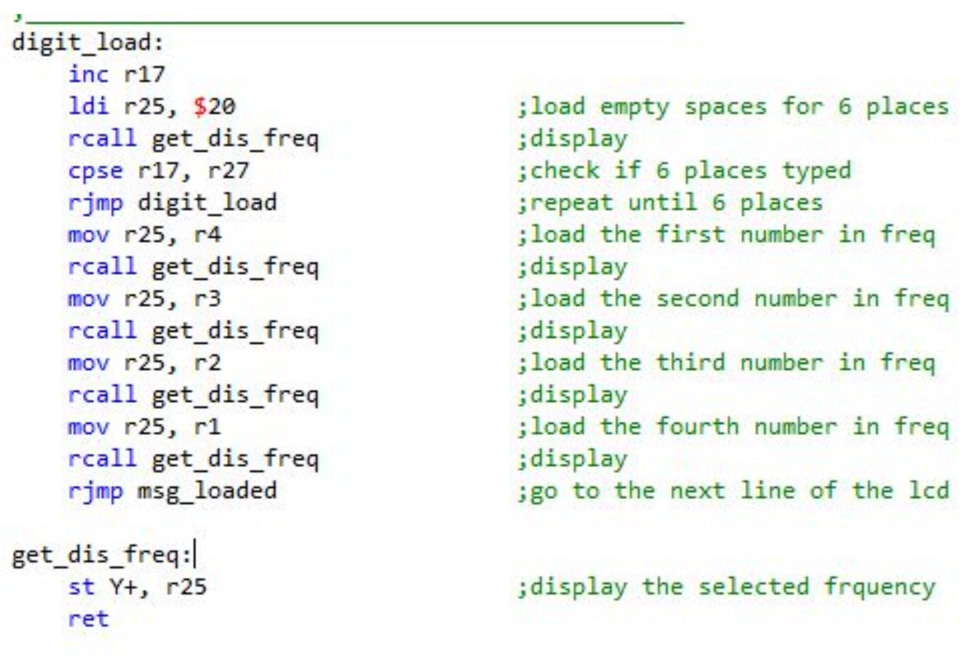


```
unpack:
    push r16                        ;store the value currently in r16
    mov r2, r8                      ;make a copy of r8 in r2
    mov r4, r9                      ;make a copy of r9 in r4
    ldi r16, $0f                    ;use and function to
    and r8, r16                     ;mask the upper nibble of r8
    mov r1, r8                      ;move lower nibble to r1
    and r9, r16                     ;mask upper nibble of r9
    mov r3, r9                      ;move lower nible to r3
    com r16                         ;load with f0 to mask lower nibble
    and r2, r16                     ;mask lower nibble of r8
    swap r2                         ;switch upper and lower nibble
    and r4, r16                     ;mask lower nibble of r9
    swap r4                         ;switch upper and lower nibble
```

The corresponding ascii values for each value stored in the register is converted using table lookup.

The codes that are required for the display characters on the lcd, is given. When run the code will clear out the values on the screen by calling clr_buff. The backlight and display is configured using update_buff. The characters to be displayed on the screen are stored to y pointer when message_dsp_loop is called. When the update_buff is called all the value stored in the y pointer will be displayed on the lcd.

Since the frequency constantly being changed, the values stored in the y pointer will be changed accordingly using the piece of code shown below. whenever line 2 is called the program will branch off to display_load. Thus every time the frequency is changed, the lcd display will be updated.



```
;_______________________________________________
digit_load:
    inc r17
    ldi r25, $20                  ;load empty spaces for 6 places
    rcall get_dis_freq            ;display
    cpse r17, r27                 ;check if 6 places typed
    rjmp digit_load               ;repeat until 6 places
    mov r25, r4                   ;load the first number in freq
    rcall get_dis_freq            ;display
    mov r25, r3                   ;load the second number in freq
    rcall get_dis_freq            ;display
    mov r25, r2                   ;load the third number in freq
    rcall get_dis_freq            ;display
    mov r25, r1                   ;load the fourth number in freq
    rcall get_dis_freq            ;display
    rjmp msg_loaded               ;go to the next line of the lcd

get_dis_freq:
    st Y+, r25                    ;display the selected frquency
    ret
```