Theory

Hardware:

From the last lab, the changes made to the schematic design changed the 8 pin switch buttons to 2 4 pin ones. There is really no change in the function of the switches with this change, but it serves to separate the lower and upper nibble. Another change would be the addition of an led and a d flip flop. The purpose of that is to create a square wave with the clock that is the PA0 output.

The PA0 output will be the CLK signal that will toggle the signal output sqwave. Since each time the flip flop is triggered by the clock signal then the resulting sqwave will be ½ the frequency of the clk signal. Since each entire period of a CLK signal will be a '1' or '0' that is outputted by Q. The LED before the D flip flop will show when the value stored in the D flip flop will be toggled. The LED after will demonstrate the logic that is currently being outputted by the D flipflop. Since LEDs are active high inputs, the LEDs will be in the direction away from the pin through a 270 ohm resistor into ground. The LED after the flipflop will also be oriented the same way with the led facing in the direction of the current through a 270 ohm resistor and into the ground. Since the D flip flop needs to be configured to be a toggle flipflop the Q' of the D flip flop will just be connected to the D input of the flip flop such that whenever the clock cycles the flip flop will toggle the stored input.

The chip used in for this experiment is 74HC74 Dual D-Type Flip Flop, where pin 14 is connected to vcc and pin 7 is connected to the ground. Since only one side is used pin 1 and pin 4 is connected to vcc, such that only the clock will change the input of the flip flop. The

signal for the clock is reassigned when the LOAD pushbutton is pressed. When the button is released the signal for the clock will stop generating and the value at Q will stay the same. The D flip flop is rated to function with a minimum voltage of 2v to maximum voltage of 6v.[1] It is made up of bunch of NAND gates, Nor gates, Inverters, buffers and And gate. The flip flop requires a setup up time for clock and data before clock of 25ns when operating at close to 5 v.

A Duty cycle of the wave is the ratio of the period a wave is on over the total period times 100. The duty cycle is listed in percentages. A duty cycle of a wave is very important when considering to use Pulse Width Module method. Unlike past century, all electronics contain digital circuitry. The main advantage of analog circuits is that the amount of current going into a component could be easily changed. Thus brightness of LEDs and motor speed could be easily controlled. This is not possible in digital circuits because it has only two voltages 0v and 5v(not exact, this will vary slightly), Thus could not be adjusted to yield certain amount of current. To solve this problem, Pulse Width Module was implement. The PWM solves this problem by repeatedly turn on and off the circuitry at certain interval, this interval of turning on and off is termed as the Duty cycle. For this lab, an LED is implemented using PWM. In order to adjust the brightness of the led, the duty cycle of the wave will be adjusted. The turning on and off a led at certain duty cycle does not dim the led, the led will turn on with the same voltage and current. But it will trick a human eye into believing the brightness of the LED is changed. A human eye works by absorbing the photons emitted, more amount of photon equals brighter light and vice versa. By flickering the led at a high

[1] courtesy datasheet

speed will give an illusion as the led is continuously on, and the amount of photons absorbed by the retina will be higher or lower depending on the duty cycle and frequency.

Program Code:

For this lab assigment the subroutines can be called using rcall. In order to use the subroutines, stack pointers have to be initialized. This is done using the segment of code in figure below.

```
//initizling the stack pointer
ldi r16, LOW(RAMEND)            ;load SPL with low byte of
out SPL, r16                    ;RAMEND adress
ldi r16, HIGH(RAMEND)           ;load SPH with low byte of
out SPH, r16                    ;RAMEND adress
```

Stack pointers are highly advantageous for coding in assembly. A stack is a consecutive block of data memory allocated by the programmer. This block of memory can be use both by the microcontroller internal control as well as the programmer to store data temporarily. The stack operates with a Last In First Out (LIFO) mechanism, i.e the last thing store on the stack is the first thing to be retrieved from the stack.[2] The stack pointer is located in the SRAM, the stack pointer has 1024 bytes of memory, thus the entire SRAM can be utilized as a stack. After initializing stack pointers, the program is allowed to use push, pop, rcall etc. Failure to initialize stack pointers while using push, pop, rcall will result in an error while building the program.

The first program utilizes the inputs from the lower nibble of the DIP switches to generate waveform with the duty cycle specified in the switches. Since the instructions indicated the frequency of the signal to be 1Khz, the waveform generated had to have a period of 1ms or 1000 us(1/1khz = 1ms). A waveform has be in logic 1 for the amount of time

[2] courtesy http://www.avr-tutorials.com/general/avr-microcontroller-stack-operation-and-stack-pointer

specified by the duty cycle. For example, if the duty cycle was inputted to be 20 percent, The waveform should be in logic 1 position for 20 percent of the time ie, 200us, and be in logic 0 position for the rest ie, 800us. In total this waveform will have a period of 1000us. This is done by utilizing delays. As shown in figure 2, the input value of the switch is loaded to r21, and decimal value 33 is loaded into r20. The r20 will be the delay of the inner loop. Since the inner_loop is 3 clock cycles long, the value of 33 *3 will give 99, which is almost 100. This loop is repeated for the duty cycle requested in the switches. For example if the switch value is 4, the delay for logic 1 will be nearly 400 clock cycles. delay_off subroutines does the same function but for the value 10 minus the switch value.

```
//read values of switch
read_switch:
    in r18, PIND                ;input the switch values to r18
//  ldi r16, 0                  ;load with 0 to compare with swtich
    CPI r18, 0                  ;check weather r18 is 0
    BREQ clear                  ;jump to clear if equal
    ldi r16, $f6                    ;load r16 with 90 in hex, to check if its 9+
    add r16,r18                 ;if there is a negative value then
    brcs clear                  ;output 0 for anything greater than 9

    rcall hex_7seg              ;go to hex7seg and return
    rcall duty_pwm              ;go to dutypwm and return

//output the pwm singal
duty_pwm:
    SBIC PINC, 7                ;check if load button is pressed
    rjmp main_loop              ;if pressed restart
    SBI PORTA, 0                ;set port a to high
    rcall delay_on              ;turn on the led for cetain period
    CBI PORTA, 0                ;set port a to low
    rcall delay_off             ;turn off the led for certain period
    rjmp duty_pwm               ;output the signal until button press
```

figure 1

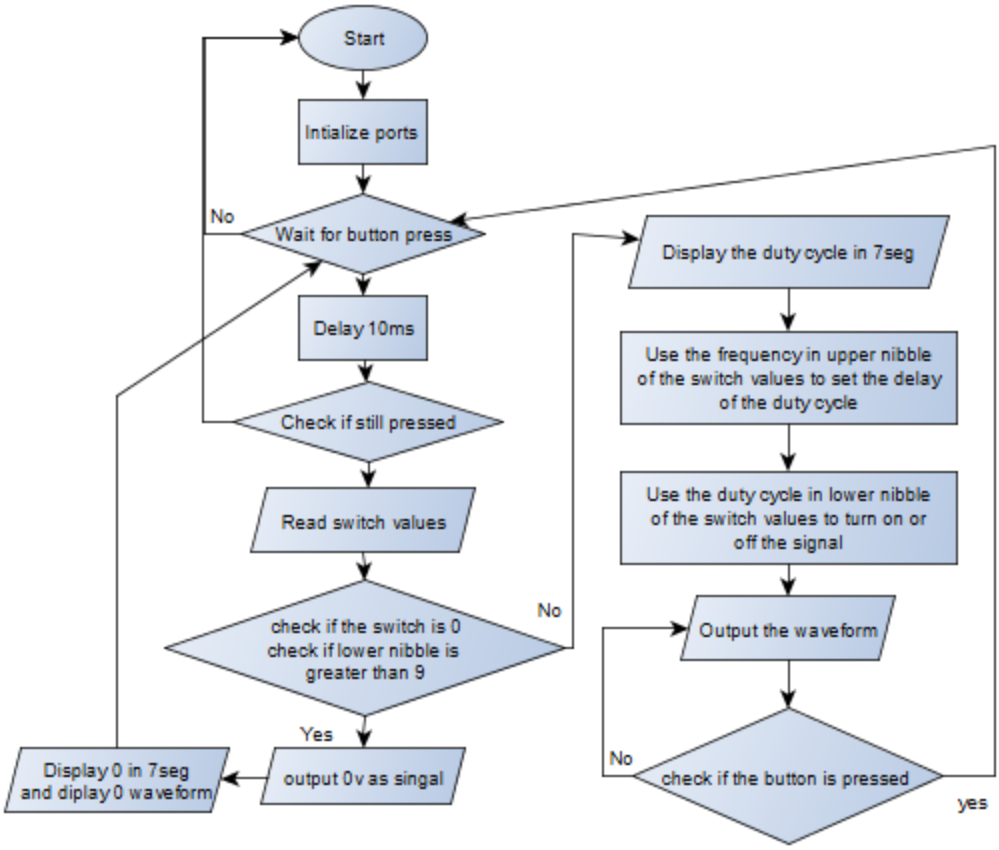
```
delay_on:
    ldi r20, 33             ;load r20 with 100
    mov r21, r18                ;load r21 with switch values
    rjmp inner_loop             ;delay for r18*100 cycles

//turn off the signal for 10 minus switch value
delay_off:
    ldi r20, 33                 ;load r20 with 100
    ldi r21, 10                 ;load r21 with 10
    sub r21, r18                ;subract 10 - switch values
    rjmp inner_loop             ;delay for r21*100 cycles

delay:
    ldi r20, inner              ;set r20 to 240
    ldi r21, outer              ;   set r21 to 13
inner_loop:
    dec r20                     ;decrements 240
    brne inner_loop             ;repeat until r20 is 0
outer_loop:
    ldi r20, inner              ;reset the r20 to 240
    dec r21                     ;decrement for 13 cycles
    brne inner_loop             ;repeat until r21 is 0
    ldi r21, outer              ;reset r21 for next delay
    ret                         ;return
```

figure 2

The input values of the switch will be in binary, 0-9 will indicate the percentages of the duty cycle. Any input values above 9 will be indicated by displaying 0 in 7seg and display no waveform in this program. To check whether the input value is greater than 9, the program will add 6 to the value. Thus anything above 9 will add up to more than 15 will be set the carry flag, thus whenever the carry flag is on when adding, the program will output no waveform.

The Duty cycle of each waveform, for convenience will be indicated in the 7seg LED display. From previous lab the Push button switches remain, The program will utilizes one of the pbsw as the load button. The program will read the switch values only when the pushbutton is pressed, and will display the same waveform continuously until the next press. Same as previous lab, this Pushbutton also has a debounce delay.

The second program of this lab, requires to use the upper nibble of the DIP switch to set the value of the frequency. The code required to do this function will be added on to the previous program, thus all the function will remain the same, except the adjustment of the frequency. unlike the duty cycle, the frequency will have 15 different values. Thus for every different switch input the output waveform will have different frequency. A general structure of how the program works is given in the below flowchart.



The frequency of the wave is controls the amount of delay, ie the length of the waveform. with each value the frequency changes. when the input is 15 in the upper nibble, the frequency was found to be 1.87 Khz. As the frequency increases the time period decreases, but the duty cycle remain constant.