**Project specifications:**

* Objective: generate a square wave based on user input.
* The user will be prompted to enter the frequency and duty cycle of the square signal on the LCD screen.
* The user will use a 4x4 keypad to enter the frequency and duty cycle.
* The user input will be echoed (displayed) on the screen.
* Invalid keys will be filtered (ie if user enters 123A then the screen will only display 123 and the A is disregarded, and the user will be allowed to continue entering values).
* Don't allow the user to enter an out of range number (i.e. Input can't exceed the max for frequency and duty cycle. Max for duty cycle is 100. Max for frequency is be determined by you).
* Convert the user input to a number "on the fly" (ie update the number as the user is inputting digits and don't store the input into a string array first).
* Extra credit: allow the user to "backspace" (delete input 1 character a time).
* The LCD 4-bit mode is preferred, but 8-bit mode is OK.
* The program should repeat until user decides to quit.
* You should build your project in Simulation IDE and use Microchip Studio for programming.
* Use the oscilloscope in Simulation IDE to confirm the wave generation.

**Process:**

1. Prompt the user to enter frequency
2. Convert the input to a number "on the fly".
3. Prompt the user to enter duty cycle
4. Convert the input to a number "on the fly"
5. Calculate period and duty cycle count.
6. Use PWM to generate signal
7. Repeat step 2, until user quits.

**Algorithms:**

Process 1&3: Prompt the user to enter frequency & duty cycle.

* Display a message indicating what the app does, e.g. "square wave generator"
* Then prompt the user to enter frequency in Hz and duty cycle as a percentage
* Remember that the LCD is 2x16 display, so make sure that your message doesn't run over 16 characters and that you leave enough space for the user input.
* Designate a key for ENTER (choose one of the special keys: \*,#,A,B,C,D)
* Designate a key for Quit (another special key)
* Inform the user of the ENTER, QUIT, MAX values of freq and duty cycle

Process 2&4: Convert the input to a number "on the fly"

1. Initialize the number: num = 0
2. Read key = single digit input. Debounce input.
3. If key = ENTER then jump to stop
4. else if key = QUIT then jump to terminate program
5. else if: key = forbidden character jump to step 2 (filter/ignore bad chars)
6. else: {echo (display) key to LCD
7. convert key to a digit: d = k - 0x30
8. Update the number: num = num\*10 + d}
9. If num > max then jump to step 1
10. Jump to step 2

Process 5. Calculate period and duty cycle count.

* It is best to use one of the 16 bit timers (1,3,4,5). You may choose any channel (OCA, OCB, OCC)
* period count = CPU\_clock/(signal freq\*prescaler) = 16MHz/(signal freq\*prescaler)
* Choose a prescaler to guarantee that period count < 65535:
  1. if        16M/(signal freq) < 65535, then prescaler = 1
  2. else if 16M/(signal freq)/8 < 65535, then prescaler = 2
  3. else if 16M/(signal freq)/64 < 65535, then prescaler = 3
  4. else if 16M/(signal freq)/256 < 65535, then prescaler = 4
  5. else    16M/(signal freq)/1024 < 65535, then prescaler = 5
* Extra credit: For accurate signal generation check if there is a remainder in the division: ie 16M%(signal freq \* prescaler) > 0. Avoid prescalers that result in fractions. However, be careful that for some frequencies this unavoidable (e.g. 3Hz)
* duty count = period count \* duty/100
* To get a full duty cycle in 1% increments: period count >= 100
* To get a duty cyle in 5% increments: period count >= 20
* To get a duty cyle in 10% increments: period count >= 10

Process 6. Setup the PWM

* Check the notes on how to use the PWM. Fast PWM allows for highest frequency

**Submission:**

Submit **1 zip** file that contains

* a report document,
* a simulation IDE file (.sim)
* a C program (.C)
* a hex file for the simulation (.hex)

**Report:**

In your report include the following (don't do a full report format):

* Indicate which keys are used for control (backspace, carriage return, quit).
* Schematic showing all components and their pin connections (ie screenshot you simlation IDE). The circuit may be too big to fit on a page, so split it into separate images. Label wires if needed (e.g if they're cut-off).
* Derive:
  + MAX frequency that can be produced accurately.
  + The max duty cycle that can be detected at the max frequency
  + The max frequency at which duty cycle is properly detected in 10% increments
  + The max frequency at which duty cycle is properly detected in 1% increments.
* Code:
  + Code will not be graded if you don't follow instruction below:
    - INDENT code
    - DON'T double space the code or use a font larger that 11pts
    - DON'T use running comment lines (that continue on the next line)
* Test results (image screenshots):  
  + MAX freq detected properly
  + MAX freq at which 1%, 99% duty is produced properly
  + MAX freq at which 5%, 95% duty is produced properly
  + MAX freq at which 10%, 90% duty is produced properly
  + Screenshot should clearly show the scope screen.

**Project 2 Report**

INSTRUCTIONS:

* # is DELETE
* \* is ENTER
* A is TERMINATE
* PRESSING 1 AT THE START 🡪 Automatically sets freq/duty = max

For example, pressing 1 in the 0th index sets 100,000 Hz. Otherwise, 0 must be pressed.

All valid frequencies will work properly and be detected up to 100 kHz, which is the limit.

* A computer screen shot of a computer

  Description automatically generated
* A computer screen shot of a number pad

  Description automatically generated
* A computer screen shot of a computer

  Description automatically generated

A screenshot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

A computer screen shot of a computer

Description automatically generated

Max Freq 100 kHz @ 1% Duty

A computer screen shot of a circuit board

Description automatically generated

1% Freq of Max (1 kHz) @ 99% Duty

A computer screen shot of a blue circuit board

Description automatically generated

Max Freq 100 kHz @ 5% Duty

A computer screen shot of a circuit board

Description automatically generated

5% Freq of Max (5 kHz) @ 95% Duty

A computer screen shot of a blue circuit board

Description automatically generated

Max Freq 100 kHz @ 10% Duty

A computer screen shot of a circuit board

Description automatically generated

10% Freq of Max (10 kHz) @ 90% Duty

A computer screen shot of a blue circuit board

Description automatically generated

More Examples

A computer screen shot of a circuit board

Description automatically generated

A computer screen shot of a blue circuit board

Description automatically generated

A computer screen shot of a blue circuit board

Description automatically generated

**Code below**

Code

// Project 2

#define *F\_CPU* 16000000L //16MHz

#include <avr/io.h>

#include <util/delay.h>

#include <stdint.h>

void setupLCD(void)

{

DDRF = 0xFF; //LCD 8bit data

DDRK = 0x03; //PTK0= En, PTK1=RS

char2LCD(0x38, 0); // set 8-bit data, 2-line display, 5x8 font

char2LCD(0x0F, 0); // turn on display, cursor, blinking

char2LCD(0x06, 0); // move cursor right

char2LCD(0x01, 0); // clear screen, move cursor to home

}

char keypad()

{

char col[4] = {0x70, 0xB0, 0xD0, 0xE0};

char key[4][4] = {

{'1','2','3','A'},

{'4','5','6','B'},

{'7','8','9','C'},

{'\*','0','#','D'}};

int r = -1;

DDRA = 0xF0;

for (int c = 0; c < 4; c++) // enable each column 1 at a time

{

PORTA = col[c];

if (PINA != 0x0F)

{

*\_delay\_ms*(10); //debounce

switch (PINA & 0x0F)

{

case(0b0111): r = 0; break;

case(0b1011): r = 1; break;

case(0b1101): r = 2; break;

case(0b1110): r = 3; break;

default: break;

}

if(r != -1)

{

return (key[r][c]);

}

}

}

return ' ';

}

void char2LCD (unsigned char data, unsigned char IR\_DR)

{ PORTK = IR\_DR; // set instr or character

PORTK = IR\_DR|0x01; // pull E signal to high

PORTF = data;

PORTK = IR\_DR;

}

void str2LCD (unsigned char \*ptr)

{

while (\*ptr) //ascii-Z strings (null-terminated)

{

char2LCD(\*ptr, 2);

// \_delay\_ms(500);

ptr++;

}

}

// Inputs a char at a specific area of a string to the LED

void input\_char(*int8\_t* character, *uint8\_t* \*str, *int8\_t* len)

{

*int8\_t* i;

for (i = len - 1; i >= 0; i--){

str[i] = character;

}

}

// Converts stuff into arrays/strings for LCD output

void output\_LCD(*uint32\_t* num, *uint8\_t* \*str, *int8\_t* len)

{

*int8\_t* i;

for (i = len -1; i >= 0; i--){

str[i] = (num % 10) + 0x30;

num /= 10;

}

}

void add\_digit(*uint32\_t* \*num, *uint32\_t* \*mul, *uint8\_t* key){

\*num += (*uint32\_t*) (key - 0x30) \* (\*mul);

\*mul /= 10;

}

void delete\_digit(*uint32\_t* \*num, *uint32\_t* \*mul){

\*mul \*= 10;

\*num /= \*mul;

\*num \*= \*mul;

}

*int32\_t* pwm\_settings(*uint32\_t* \*signal\_freq){

*int32\_t* TOP;

const *uint16\_t* U16\_MAX = 65535; // 16 bit limit

TCCR0A = 0b00100011;

// Extra Credit

if((*F\_CPU*/ \*signal\_freq)/1024 < U16\_MAX && !(*F\_CPU*%( \*signal\_freq\*1024) > 0)){

TOP = *F\_CPU*/( \*signal\_freq\*1024);

TCCR0B = 0b00001101;

}

else if((*F\_CPU*/ \*signal\_freq)/256 < U16\_MAX && !(*F\_CPU*%( \*signal\_freq\*256) > 0)){

TOP = *F\_CPU*/( \*signal\_freq\*256);

TCCR0B = 0b00001100;

}

else if((*F\_CPU*/ \*signal\_freq)/64 < U16\_MAX && !(*F\_CPU*%( \*signal\_freq\*64) > 0)){

TOP = *F\_CPU*/( \*signal\_freq\*64);

TCCR0B = 0b00001011;

}

else if((*F\_CPU*/ \*signal\_freq)/8 < U16\_MAX && !(*F\_CPU*%( \*signal\_freq\*8) > 0)){

TOP = *F\_CPU*/( \*signal\_freq\*8);

TCCR0B = 0b00001010;

}

else if(*F\_CPU*/ \*signal\_freq < U16\_MAX && !(*F\_CPU*% \*signal\_freq > 0)){

TOP = *F\_CPU*/ \*signal\_freq;

TCCR0B = 0b00001001;

}

// Fast PWM

else

{

if((*F\_CPU*/ \*signal\_freq)/1024 < U16\_MAX)

{

TOP = *F\_CPU*/( \*signal\_freq\*1024);

TCCR0B = 0b00001101;

}

else if((*F\_CPU*/ \*signal\_freq)/256 < U16\_MAX)

{

TOP = *F\_CPU*/( \*signal\_freq\*256);

TCCR0B = 0b00001100;

}

else if((*F\_CPU*/ \*signal\_freq)/64 < U16\_MAX)

{

TOP = *F\_CPU*/(\*signal\_freq\*64);

TCCR0B = 0b00001011;

}

else if((*F\_CPU*/ \*signal\_freq)/8 < U16\_MAX)

{

TOP = *F\_CPU*/(\*signal\_freq\*8);

TCCR0B = 0b00001010;

}

else

{

TOP = *F\_CPU*/ \*signal\_freq;

TCCR0B = 0b00001001;

}

}

return TOP;

}

int main(void){

DDRG |= 0x20; // configure OC0B pin for PWM output (PG5 is the built-in output pin)

setupLCD();

// Display Intro

str2LCD(" Square Wave ");

char2LCD(0XC0,0);

str2LCD(" Generator ");

*\_delay\_ms*(1000);

// Display Instructions

char2LCD(0X01,0);

str2LCD(" ENTER: \* ");

char2LCD(0XC0,0);

str2LCD(" DELETE: # ");

*\_delay\_ms*(1000);

// Display Max Freq.

char2LCD(0X01,0);

str2LCD(" Stop: A ");

char2LCD(0XC0,0);

str2LCD("Max Freq = 10kHz");

*\_delay\_ms*(1000);

// Input Freq.

char2LCD(0X01,0);

str2LCD("Enter Frequency:");

char2LCD(0XC0,0);

str2LCD(" Hz ");

*\_delay\_ms*(1000);

*uint8\_t* \*freq\_Hz = " Hz", \*duty\_percent = " % ";

*uint32\_t* signal\_freq, duty\_cycle;

// User input logic for freq

*int8\_t* flag = 0;

while(flag != 1){

char2LCD(0XC0,0);

signal\_freq = 0;

*uint8\_t* \*string\_build = " ";

*int8\_t* enter\_flag = 0;

*uint8\_t* current\_char = 0xC0;

*uint8\_t* key = ' ';

*uint32\_t* mul = 100000;

current\_char++;

for(*int8\_t* i = 0; i <= 6 && enter\_flag != 1; string\_build[0] = " "){

*\_delay\_ms*(100);

key = ' ';

while(key == ' '||key == 'B'||key == 'C'||key == 'D'){

key = keypad();

}

if(key == '\*'){

enter\_flag = 1;

}

else if(key == '1' && i == 0){

enter\_flag = 1;

signal\_freq = 100000;

}

else if(key == '#'){

if(i != 0){

i--;

current\_char--;

char2LCD(current\_char-1,0);

input\_char(' ', string\_build, 1);

str2LCD(string\_build);

char2LCD(current\_char-1,0);

delete\_digit(&signal\_freq, &mul);

}

}

else if(key == 'A'){

char2LCD(0X01,0);

str2LCD(" ");

char2LCD(0XC0,0);

str2LCD(" ");

*exit*(0);

}

else if(key != '0' && i == 0){

}

// Add valid key input to LED

else if(i != 6){

input\_char(key, string\_build, 1);

str2LCD(string\_build);

char2LCD(current\_char,0);

add\_digit(&signal\_freq, &mul, key);

current\_char++;

i++;

}

else{

// Do nothing in edge cases

}

}

if(signal\_freq <= 100000){

flag = 1;

}

}

output\_LCD(signal\_freq, freq\_Hz, 6);

// Max Duty Cycle

char2LCD(0X01,0);

str2LCD("Max Duty Cycle =");

char2LCD(0XC0,0);

str2LCD(" 100 % ");

*\_delay\_ms*(1000);

char2LCD(0X01,0);

// Duty Cycle Input Logic

str2LCD("Enter DutyCycle:");

char2LCD(0XC0,0);

str2LCD(" % ");

flag = 0;

while(flag != 1){

char2LCD(0XC0,0);

*uint8\_t* \*string\_build = " ";

*int8\_t* enter\_flag = 0;

*uint8\_t* current\_char = 0xC0;

*uint8\_t* key = ' ';

*uint32\_t* mul = 100;

current\_char++;

duty\_cycle = 0;

for(*int8\_t* i = 0; i <= 3 && enter\_flag != 1; string\_build[0] = " "){

*\_delay\_ms*(100);

key = ' ';

while(key == ' '||key == 'B'||key == 'C'||key == 'D'){

key = keypad();

}

if(key == '\*'){

enter\_flag = 1;

}

else if(key == '1' && i == 0){

enter\_flag = 1;

duty\_cycle = 100;

}

else if(key == '#'){

if(i != 0){

i--;

current\_char--;

char2LCD(current\_char-1,0);

input\_char(' ', string\_build, 1);

str2LCD(string\_build);

char2LCD(current\_char-1,0);

delete\_digit(&duty\_cycle, &mul);

}

}

else if(key == 'A'){

char2LCD(0X01,0);

str2LCD(" ");

char2LCD(0XC0,0);

str2LCD(" ");

*exit*(0);

}

else if(key != '0' && i == 0){

}

else if(i != 3){

input\_char(key, string\_build, 1);

str2LCD(string\_build);

char2LCD(current\_char,0);

add\_digit(&duty\_cycle, &mul, key);

current\_char++;

i++;

}

else{

// Do nothing in edge cases

}

}

if(duty\_cycle <= 100){

flag = 1;

}

}

// Display Final Values

output\_LCD(duty\_cycle, duty\_percent, 3);

*\_delay\_ms*(100);

char2LCD(0x01,0);

str2LCD("Freq = ");

str2LCD(freq\_Hz);

char2LCD(0xC0,0);

str2LCD("Duty = ");

str2LCD(duty\_percent);

*int32\_t* TOP = pwm\_settings(&signal\_freq);

*int32\_t* duty\_high = TOP\*duty\_cycle/100;

TCNT0 = 0; // force TCNT0 to count up from 0

OCR0A = TOP-1;

OCR0B = duty\_high;

while(1);

}