```
/*
 * synth1.c
 *
 * Created: 2023-03-30 13:03:00
 * Author : se6282ho-s
 */
#define F_CPU 16000000UL
#include <avr/io.h>
#include <avr/interrupt.h>
#include <math.h>
#include <avr/delay.h>

uint32_t phAcc = 0;                    // phase accumulator / counter counts up and rolls over to 0
uint8_t dacVal = 0;                    // data to send to DAC
volatile uint8_t sendSample = 0;  // flag to send a sample to the DAC when interrupt occurs

float fOut = 1000;                              // target frequency to generate in Hz
volatile uint32_t tuningWord = 0; // DDS tuning word for target frequency
uint8_t button_1_state = 1;
uint8_t button_2_state = 1;


#define bitRead(value, bit) (((value) >> (bit)) & 0x01);



float not_C = 65.40639;
float not_Css = 69.29566;
float not_D = 73.41619;
float not_Dss = 77.78175;
float not_E = 82.40689;
float not_F = 87.30706;
float not_Fss = 92.49861;
float not_G = 97.99886;
float not_Gss = 103.8262;
float not_A = 110.0000;
float not_Ass = 116.5409;
float not_B = 123.4708;

uint8_t octav_Changer = 0;
uint8_t octav_Higher_WasPressed = 0;
uint8_t octav_Lower_WasPressed = 0;
#define DEBOUNCE_TIME 3000
uint8_t oct = 0;
```

```
int wave_Select = 0;
uint8_t wave_select_WasPressed = 0;
uint8_t wave_Form = 0;

uint8_t LUT[] = {128, 131, 134, 137, 140, 143, 146, 149, 152, 155, 158, 162, 165, 167, 170, 173,
176, 179, 182, 185, 188, 190, 193, 196, 198, 201, 203, 206, 208, 211, 213, 215,
    218, 220, 222, 224, 226, 228, 230, 232, 234, 235, 237, 238, 240, 241, 243, 244, 245,
246, 248, 249, 250, 250, 251, 252, 253, 253, 254, 254, 254, 255, 255, 255,
    255, 255, 255, 255, 254, 254, 254, 253, 253, 252, 251, 250, 250, 249, 248, 246, 245,
244, 243, 241, 240, 238, 237, 235, 234, 232, 230, 228, 226, 224, 222, 220,
    218, 215, 213, 211, 208, 206, 203, 201, 198, 196, 193, 190, 188, 185, 182, 179, 176,
173, 170, 167, 165, 162, 158, 155, 152, 149, 146, 143, 140, 137, 134, 131,
    128, 124, 121, 118, 115, 112, 109, 106, 103, 100, 97, 93, 90, 88, 85, 82, 79, 76, 73, 70,
67, 65, 62, 59, 57, 54, 52, 49, 47, 44, 42, 40,
    37, 35, 33, 31, 29, 27, 25, 23, 21, 20, 18, 17, 15, 14, 12, 11, 10, 9, 7, 6, 5, 5, 4, 3, 2, 2, 1,
1, 1, 0, 0, 0,
    0, 0, 0, 0, 1, 1, 1, 2, 2, 3, 4, 5, 5, 6, 7, 9, 10, 11, 12, 14, 15, 17, 18, 20, 21, 23, 25, 27, 29,
31, 33, 35,
    37, 40, 42, 44, 47, 49, 52, 54, 57, 59, 62, 65, 67, 70, 73, 76, 79, 82, 85, 88, 90, 93, 97,
100, 103, 106, 109, 112, 115, 118, 121, 124
};

uint8_t waveDebounce (void){
    if(!(PINA & (1<<PINA3))){
        _delay_us(DEBOUNCE_TIME);
        if (!(PINA & (1<<PINA3)))
        {
            return(1);
        }
        return 0;
    }
}


uint8_t debounceHigher (void){
    if(!(PINA & (1<<PINA5))){
        _delay_us(DEBOUNCE_TIME);
        if (!(PINA & (1<<PINA5)))
        {
            return(1);
        }
         return 0;
    }
```

```
}

uint8_t debounceLower (void){
        if(!(PINA & (1<<PINA4))){
                _delay_us(DEBOUNCE_TIME);
                if (!(PINA & (1<<PINA4)))
                {
                        return(1);
                }
                return 0;
        }
}


int main(void)
{

        DDRD = 0b00000000;
        PORTD = 0b11111111;
        DDRC = 0b00000000;
        PORTC = 0b00001111;
        DDRA = 0b00000000;
        PORTA = 0b11111000;

        DDRB |= 0xff;
                //set timer1 interrupt for 9060 Hz @ 16 MHz clock, no prescale
                cli();
                TCCR1A = 0;    // clear register
                TCCR1B = 0;    // clear register
                TCNT1  = 0;    // initialize counter value to 0
                OCR1A = 1766;  // 16MHz / 1766  = 9060 Hz
                TCCR1B |= (1 << WGM12) | (1 << CS10);  // turn on CTC mode, Set CS10 for 1
prescaler
                TIMSK1 |= (1 << OCIE1A);               // enable timer compare interrupt
                sei();


    /* Replace with your application code */
    while (1)
    {

        if (debounceHigher()) {
                if ((octav_Changer < 5) && (octav_Higher_WasPressed == 0)) {
                        octav_Changer++;
                        oct = pow(2,octav_Changer);
                        octav_Higher_WasPressed = 1;
                }
        }   else {
                        octav_Higher_WasPressed = 0;
                }

                if (debounceLower()) {
                        if ((octav_Changer > 0) && (octav_Lower_WasPressed == 0)) {
                            octav_Changer--;
                            oct = pow(2,octav_Changer);
                            octav_Lower_WasPressed = 1;
                        }
                        }   else {
                        octav_Lower_WasPressed = 0;
                }


                 if (waveDebounce()) {
                     if (wave_select_WasPressed == 0) {
                         if (wave_Select == 3)
                         {
                             wave_Select = -1;
                         }
                            wave_Select++;
                            }



                              wave_select_WasPressed = 1;

                     }
                            else {
                     wave_select_WasPressed = 0;
                 }




        if (!(PIND & (1<<PIND6))) { // C
                fOut = not_C;
        } else if (!(PIND & (1<<PIND5))){ // D
                fOut =  not_D;
        } else if (!(PIND & (1<<PIND4))) { // E
```

```
                fOut =  not_E;
        } else if (!(PIND & (1<<PIND3))) { // F
                fOut = not_F;
        } else if (!(PIND & (1<<PIND2))) { // G
                fOut =  not_G;
        } else if (!(PIND & (1<<PIND1))) { // A
                fOut =  not_A;
        } else if (!(PIND & (1<<PIND0))) { // B
                fOut =  not_B;
        } else if (!(PIND & (1<<PIND7))) { // C#
                fOut =  not_Css;
        } else if (!(PINC & (1<<PINC0))) { // D#
                fOut =  not_Dss;
        } else if (!(PINC & (1<<PINC1))) { // F#
                fOut = not_Fss;
        } else if (!(PINA & (1<<PINA6))) { // G#
                fOut = not_Gss          ;
        } else if (!(PINA & (1<<PINA7))) { // A#
                fOut =  not_Ass;
        } else {
                fOut = 0;
        }
        fOut = fOut * oct;
        tuningWord = pow(2, 32) * fOut / 9060.0;

  }

}
ISR(TIMER1_COMPA_vect)
{
                                uint8_t count = (phAcc >> 24);      // take the 8 upper bits of the 32 bit phase accumulator as a LUT sample pointer

        if (wave_Select == 0)
        {
                 dacVal = count;                //   ramp wave:  send counter value to the DAC to generate a rising ramp
        }
        else if (wave_Select == 1)
        {
                 dacVal = (count > 127) ? 255 : 0; // square wave:  send all 0 or all 1 to the DAC for each half of a wave cycle
        }
        else if (wave_Select == 2)
        {
                dacVal = LUT[count];                //   sine wave:  send look up table sample value
to the DA
        }



                                // send calculated sine to the DAC
                                // uint8_t sineCount = (uint8_t) ((1 + sin(((2.0 * PI) / 256) * count))
* 127.5);
                                // dacVal = sineCount;

                                // send the 8 bit counter data to the DAC pins
                                PORTB = dacVal;
                                /**for (uint8_t i = 0; i <= 7; i++)
                                {
                                        PORTC |= (PINCi)
                                        digitalWrite(i , bitRead(dacVal, i));
                                }**/

                                phAcc += tuningWord; // increment phase accumulator/counter

}
```