***Coding of the clock alarm system***

#include <SevSeg.h>

#include "Button.h"

#include "AlarmTone.h"

#include "Clock.h"

#include "config.h"

const int COLON\_PIN = 13;

const int SPEAKER\_PIN = A3;

Button hourButton(A0);

Button minuteButton(A1);

Button alarmButton(A2);

AlarmTone alarmTone;

Clock clock;

SevSeg sevseg;

enum DisplayState {

  DisplayClock,

  DisplayAlarmStatus,

  DisplayAlarmTime,

  DisplayAlarmActive,

  DisplaySnooze,

};

DisplayState displayState = DisplayClock;

long lastStateChange = 0;

void changeDisplayState(DisplayState newValue) {

  displayState = newValue;

  lastStateChange = millis();

}

long millisSinceStateChange() {

  return millis() - lastStateChange;

}

void setColon(bool value) {

  digitalWrite(COLON\_PIN, value ? LOW : HIGH);

}

void displayTime() {

  DateTime now = clock.now();

  bool blinkState = now.second() % 2 == 0;

  sevseg.setNumber(now.hour() \* 100 + now.minute());

  setColon(blinkState);

}

void clockState() {

  displayTime();

  if (alarmButton.read() == Button::RELEASED && clock.alarmActive()) {

    // Read alarmButton has\_changed() to clear its state

    alarmButton.has\_changed();

    changeDisplayState(DisplayAlarmActive);

    return;

  }

  if (hourButton.pressed()) {

    clock.incrementHour();

  }

  if (minuteButton.pressed()) {

    clock.incrementMinute();

  }

  if (alarmButton.pressed()) {

    clock.toggleAlarm();

    changeDisplayState(DisplayAlarmStatus);

  }

}

void alarmStatusState() {

  setColon(false);

  sevseg.setChars(clock.alarmEnabled() ? " on" : " off");

  if (millisSinceStateChange() > ALARM\_STATUS\_DISPLAY\_TIME) {

    changeDisplayState(clock.alarmEnabled() ? DisplayAlarmTime : DisplayClock);

    return;

  }

}

void alarmTimeState() {

  DateTime alarm = clock.alarmTime();

  sevseg.setNumber(alarm.hour() \* 100 + alarm.minute(), -1);

  if (millisSinceStateChange() > ALARM\_HOUR\_DISPLAY\_TIME || alarmButton.pressed()) {

    changeDisplayState(DisplayClock);

    return;

  }

  if (hourButton.pressed()) {

    clock.incrementAlarmHour();

    lastStateChange = millis();

  }

  if (minuteButton.pressed()) {

    clock.incrementAlarmMinute();

    lastStateChange = millis();

  }

  if (alarmButton.pressed()) {

    changeDisplayState(DisplayClock);

  }

}

void alarmState() {

  displayTime();

  if (alarmButton.read() == Button::RELEASED) {

    alarmTone.play();

  }

  if (alarmButton.pressed()) {

    alarmTone.stop();

  }

  if (alarmButton.released()) {

    alarmTone.stop();

    bool longPress = alarmButton.repeat\_count() > 0;

    if (longPress) {

      clock.stopAlarm();

      changeDisplayState(DisplayClock);

    } else {

      clock.snooze();

      changeDisplayState(DisplaySnooze);

    }

  }

}

void snoozeState() {

  sevseg.setChars("\*\*\*\*");

  if (millisSinceStateChange() > SNOOZE\_DISPLAY\_TIME) {

    changeDisplayState(DisplayClock);

    return;

  }

}

void setup() {

**Serial**.begin(115200);

  clock.begin();

  hourButton.begin();

  hourButton.set\_repeat(500, 200);

  minuteButton.begin();

  minuteButton.set\_repeat(500, 200);

  alarmButton.begin();

  alarmButton.set\_repeat(1000, -1);

  alarmTone.begin(SPEAKER\_PIN);

  pinMode(COLON\_PIN, OUTPUT);

  byte digits = 4;

  byte digitPins[] = {2, 3, 4, 5};

  byte segmentPins[] = {6, 7, 8, 9, 10, 11, 12};

  bool resistorsOnSegments = false;

  bool updateWithDelays = false;

  bool leadingZeros = true;

  bool disableDecPoint = true;

  sevseg.begin(DISPLAY\_TYPE, digits, digitPins, segmentPins, resistorsOnSegments,

               updateWithDelays, leadingZeros, disableDecPoint);

  sevseg.setBrightness(90);

}

void loop() {

  sevseg.refreshDisplay();

  switch (displayState) {

    case DisplayClock:

      clockState();

      break;

    case DisplayAlarmStatus:

      alarmStatusState();

      break;

    case DisplayAlarmTime:

      alarmTimeState();

      break;

    case DisplayAlarmActive:

      alarmState();

      break;

    case DisplaySnooze:

      snoozeState();

      break;

  }

}