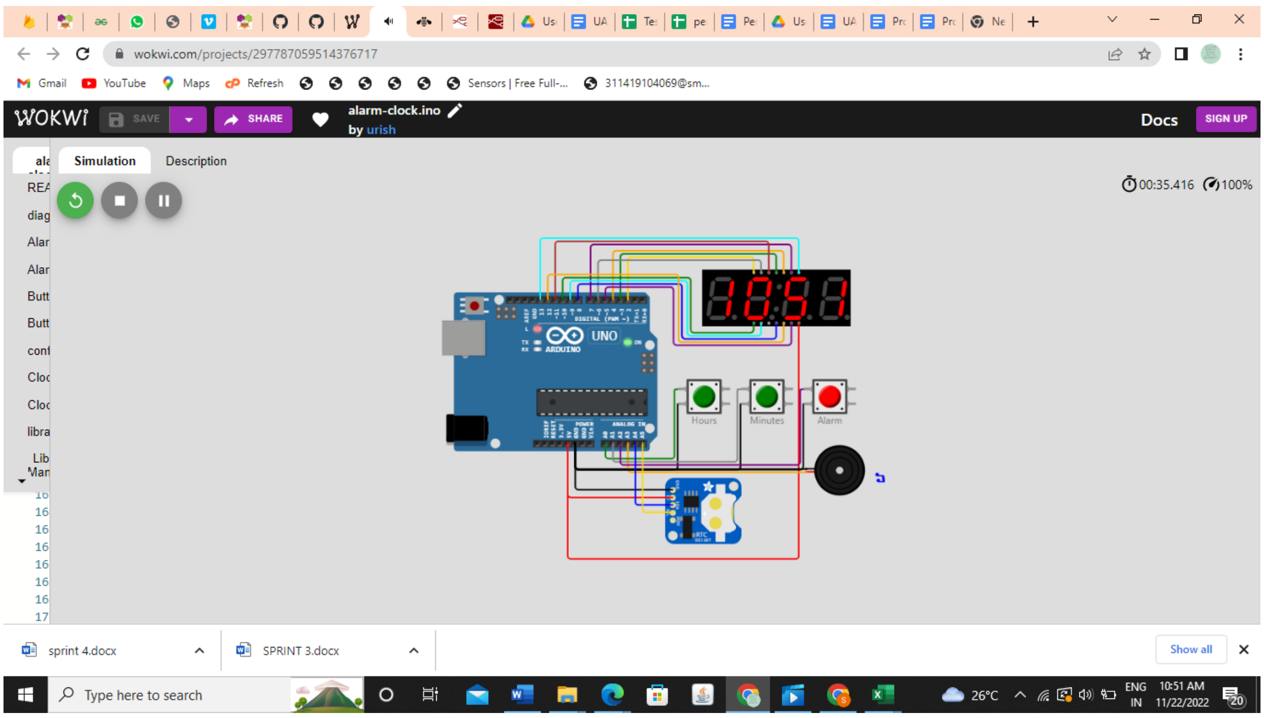
TEAM ID	**PNT2022TMID27689**
PROJECT NAME	**Personal Assistance for Seniors Who Are Self-Reliant**

AIM:

The aim of Sprint 1 is to simulate the Arduino using the python code**.**

Make necessary connection in Wokwi simulator for alarm remainder



CODE:

```
#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);
```

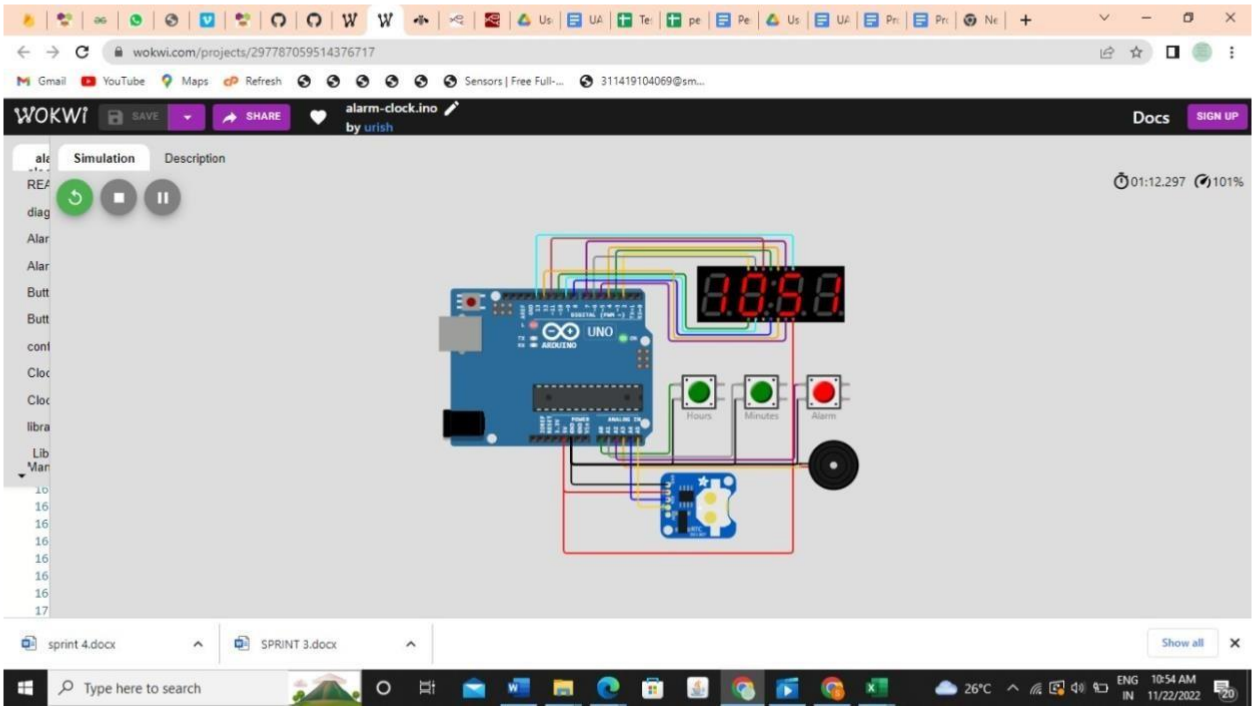
```
  sevseg.setChars(clock.alarmEnabled() ? " on" : " off");   if
(millisSinceStateChange() > ALARM_STATUS_DISPLAY_TIME) {
changeDisplayState(clock.alarmEnabled() ? DisplayAlarmTime : DisplayClock);
return;
  }
}

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);
  }
}
```

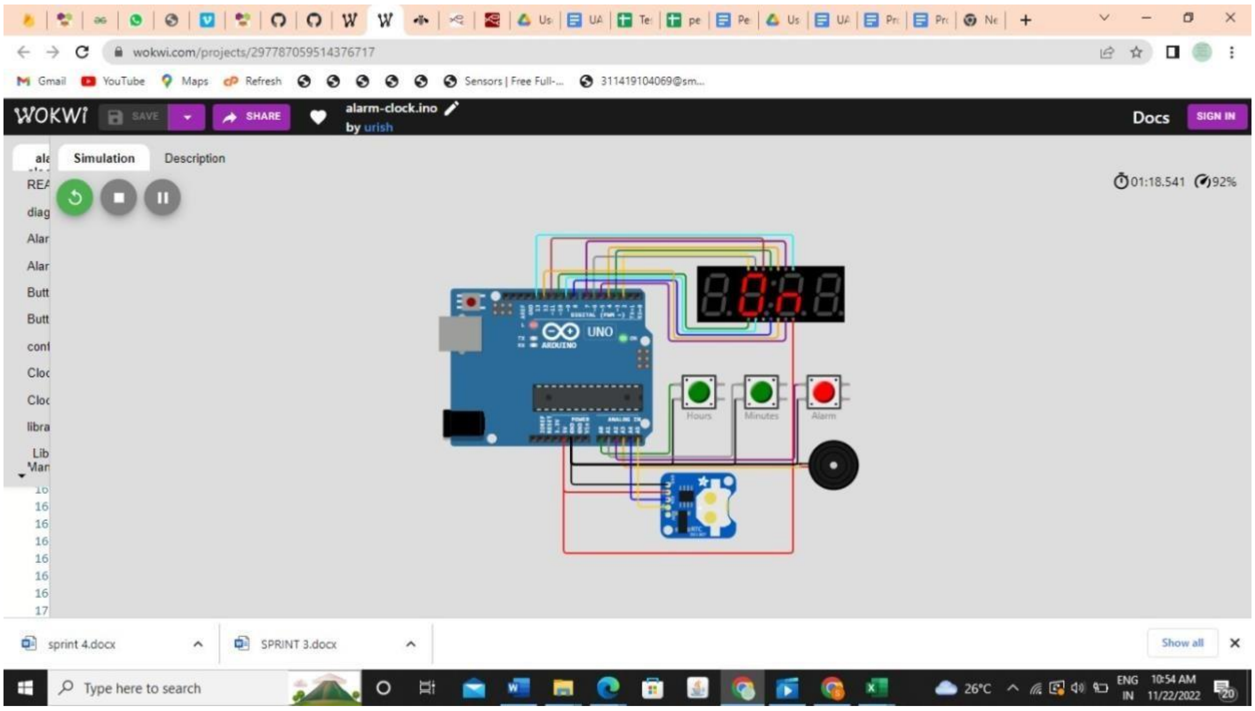
```
void alarmState() {
displayTime();

  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;
  }
}
```

Output:

At first it shows the current time





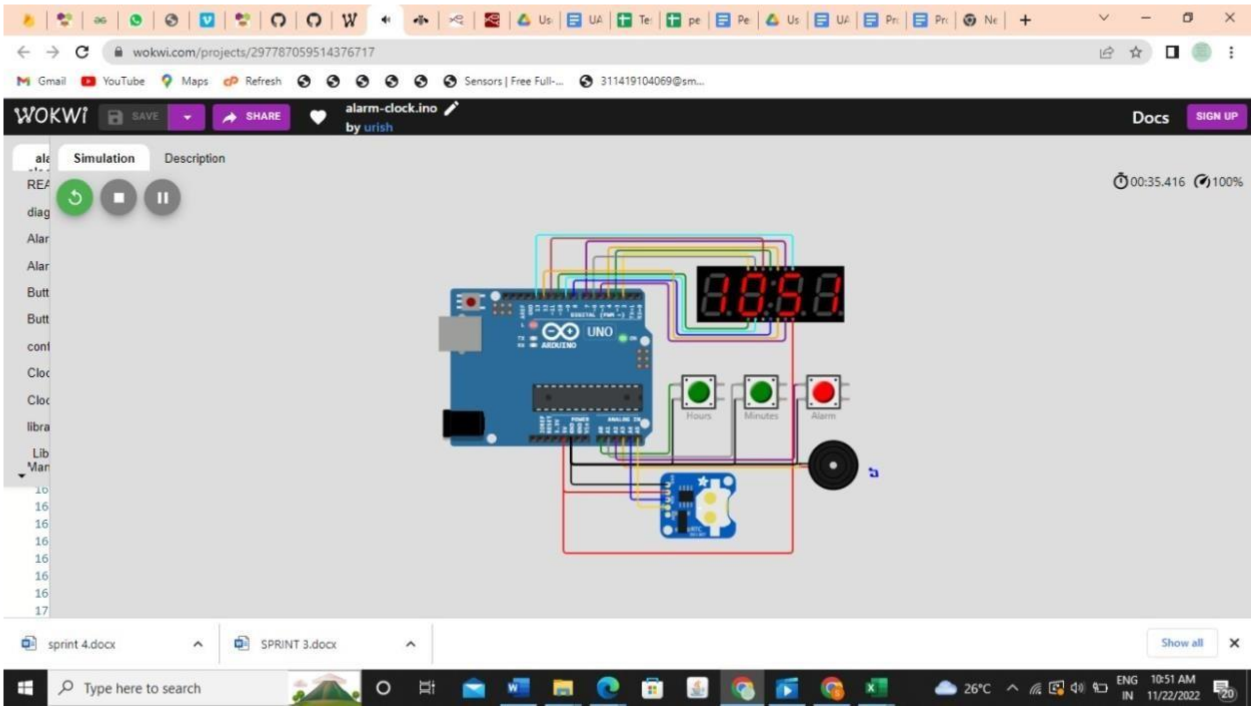
On the Alarm for setting specific time

To set the time press the minute/hour buttons. Pressing the alarm buttons enables/disables the alarm. The screen will display the alarm state by showing the word on/off

After enabling the alarm the current alarm time will be displayed for a few seconds. You can use the minute/hour button to adjust the alarm time

To finish, press the alarm button again (or) just wait for few seconds

Once we set the alarm it starts ringing



Result:

Thus, by the end of the sprint we developed the code for our alarm simulation using Arduino_UNO in Wokwi simulation