Smart Clock-IV

What is our GOAL for this CLASS?

In this class, we learned how to add timer and alarm clock features to the smart clock.

What did we ACHIEVE in the class TODAY?

- Learned to take input from users using an encoder.
- Learned to code timers and alarm clocks for smart clocks.

Which CONCEPTS/ CODING BLOCKS did we cover today?

- **Concepts**: Transmitting data, receiving data, infinite loops, sequencing of code, taking input from user using Encoder, conditional programming.
- **Coding blocks:** Serial.begin(), delay(), tone(), noTone(), constrain().

How did we DO the activities?

1. **Learn to take input from users using an encoder.**
    a. Open the wokwi simulator and replace all the files downloaded from the template.
    b. We create a function **set_value()** to store input entered by the user.
    Also, to make sure the input is valid and since the upper and lower limit for the hour, minute, and seconds differ, we use **min_val** and **max_val** variables as parameters to this function.
    This function also returns the value saved hence the return type is **int**.

```
int set_value(int min_val , int max_val){


}
```

    c. Initially, the value must be 0.

```
int value = 0;
```

    d. Reset the counter to 0.

```
counter = 0;
```

    e. Then in a **while loop**, until the user presses the push button, we print the selected input(**counter** variable) on the display.

```
while (true){
 lcd_print(0,1," ");
 lcd_print(0,1,String(counter));
}
```

    f. The **counter** variable range will keep on changing depending on the type of user input .i.e. Hour range is 0-23 and so on. Hence, we constrain it using the input parameters **min_val** and **max_val**.

```
counter = constrain(counter , min_val , max_val);
```

    g. Also, we break the **loop** if the push button is pressed.

```
button.loop();
  if (button.isPressed())
```

```
    break;
```

h. Lastly, we update the variable **value** to **counter** (the input from the encoder), return **value** and **counter** must be set back to 0 to enable the mode selection of the smart clock.

```
value = counter;
counter = 0;
return value;
```

Reference Code:

```
int set_value(int min_val , int max_val){

int value = 0;
counter = 0;

while(true){
  button.loop();
  if (button.isPressed())break;


  counter = constrain(counter , min_val , max_val);

    lcd_print(0,1,"  ");
    lcd_print(0,1,String(counter));

}

value = counter;
counter = 0;

return value;
```

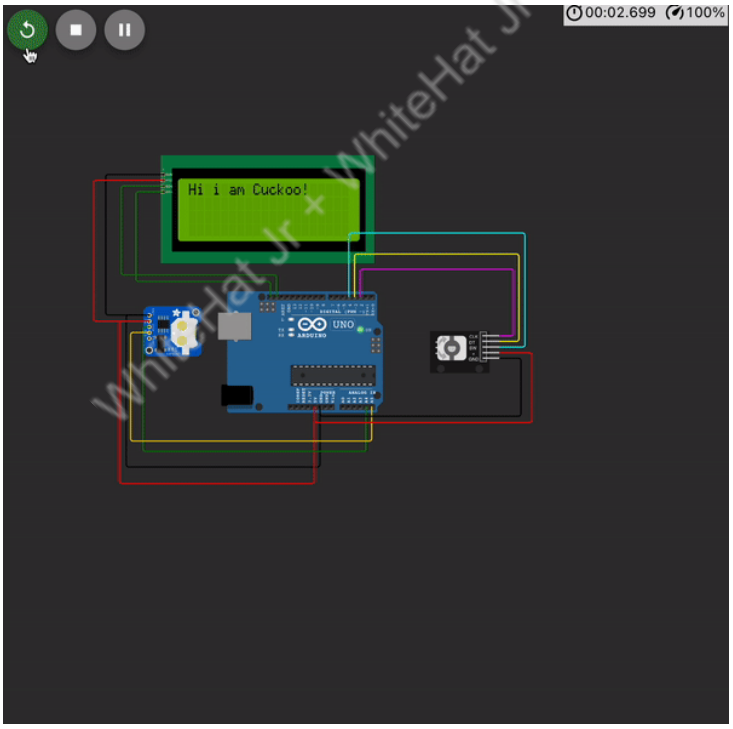
```
}
```

i. We need variables to store alarm inputs

```
// alarm variables
int alarm_hours = 0;
int alarm_minutes = 0;
```

```
void set_alarm() {
 lcd_print(0,0,"Enter hours : ");
 alarm_hours = set_value(0,23);
 Serial.print(alarm_hours);
}
```

Reference Output:



https://s3-whjr-curriculum-uploads.whjr.online/a33a8686-5334-4267-a45a-61e9155deee9.gif

j. We observed two issues here.
First, it flickers a lot. We solved it with the below step:

k. We create the variable **last_counter** and it must be different than the actual **counter** variable showing that the user has rotated the encoder and changed it and hence change the display.

```
int last_counter = -1;
```

```
if (last_counter != counter){
   lcd_print(0,1,"  ");
   lcd_print(0,1,String(counter));
   last_counter = counter;
 }
```

Then, we reset it back to -1.

```
last_counter = -1;
```

Reference Code:

```
int set_value(int min_val, int max_val) {
  int value = 0;
  int last_counter = -1;
  counter = 0;

  while (true) {
    button.loop();
    if (button.isPressed())
      break;

    counter = constrain(counter, min_val, max_val);
    if (last_counter != counter) {
      lcd_print(0, 1, "  ");
      lcd_print(0, 1, String(counter));
      last_counter = counter;
    }
  }
  value = counter;
  counter = 0;
  last_counter = -1;
  return value;
}
```

I. Next issue was with the encoder. The encoder value can be set only between 0 and 3 even after the constrain is set between 0 and 23.

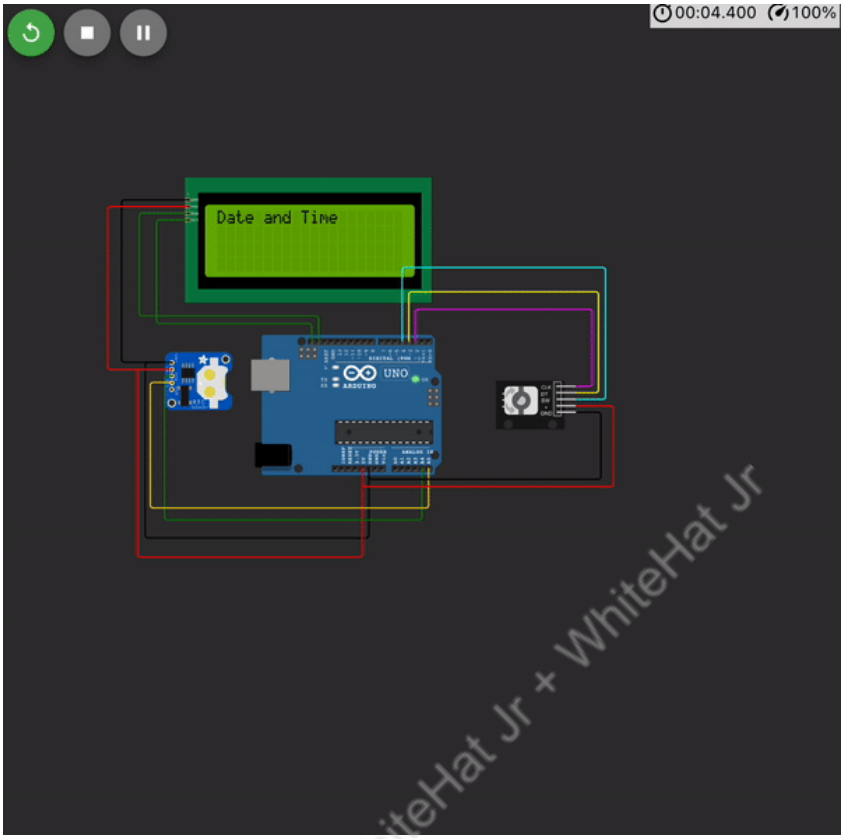
Remove the **constrain** instruction from **encoder()** and put it in **mode_selector()** as the range 0 to 3 i.e. 4 options are for mode selection.

Reference Code:

```
void encoder() {
  prev_counter = counter;
  if (digitalRead(dt)  ==  HIGH)counter++;
  else counter--;
  flag = 1;
}

void mode_selector() {
  counter = constrain(counter, 0, 3);
  if (prev_counter != counter  &&  flag  ==  1) {
    if (counter  ==  0) {
      lcd.clear();
```

Reference Output:



https://s3-whjr-curriculum-uploads.whjr.online/3b576755-e1d7-4294-8b9d-33141afd495e.gif

m. In **set_alarm(),** we accept the minutes input from the user.

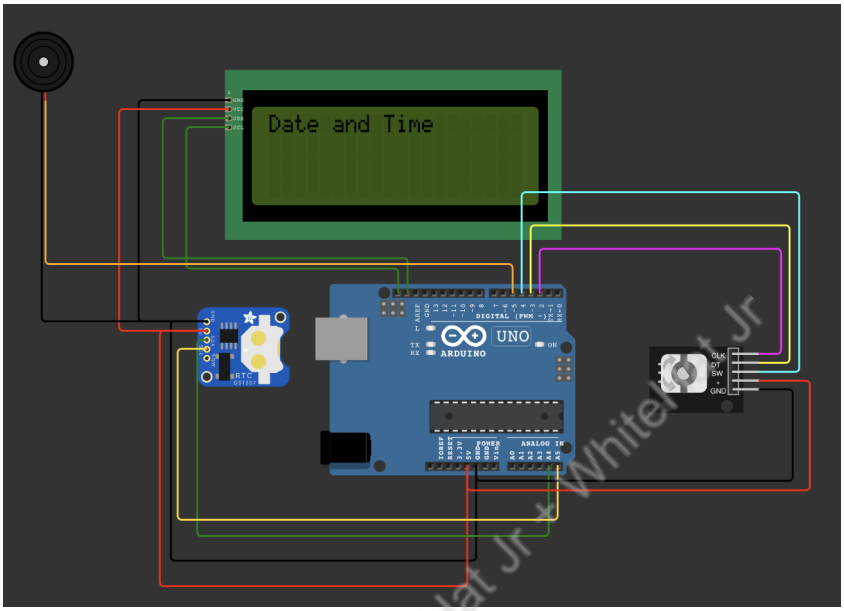
```
lcd_print(0,0,"Enter Minutes : ");
alarm_minutes = set_value(0,59);
```

n. After that, we just display a message to the user that such an alarm is set.

```
// printing data
lcd_print(0,0,"Alarm is set for ");
lcd_print(0,1,String(alarm_hours));
lcd_print(3,1," Hours and ");
lcd_print(0,2,String(alarm_minutes));
lcd_print(3,2," Minutes");
delay(5000);
```

o. We add the buzzer hardware and make its connections. **GND** of **Buzzer** to **GND** of **RTC/Arduino** and **VCC** of **Buzzer** to pin 5 of Arduino.

Reference Image:



p. Next, we create a function to play the buzzer.

```
byte buzzer_pin = 5;
```

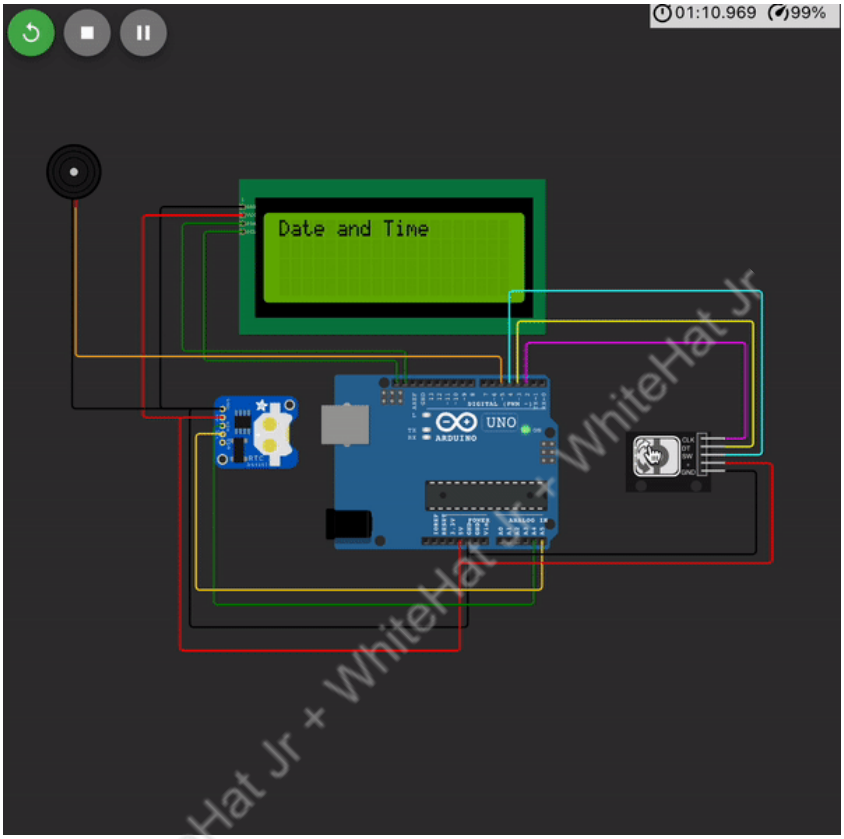
```
void play_buzzer(int frequency , int duration){
 tone(buzzer_pin , frequency);
 delay(duration);
 noTone(buzzer_pin);
}
```

q. Next, we check it with real-time in **loop()** and ring the buzzer at alarm time along with resetting the alarm variables back to 0.

```
if (alarm_minutes == minute && alarm_hours == hour){
  play_buzzer(1000 , 5000);
  // resetting variables
  alarm_hours = 0;
```

```
  alarm_minutes = 0;
}
```

Reference Output:



https://s3-whjr-curriculum-uploads.whjr.online/dbdea4df-d8ac-409b-88e5-95282a191674.gif

2. **Learn to code timer for smart clocks.**
   a. We ask the user about the timer value.

```
int countdown_minutes = 0;
 int countdown_seconds = 0;
//  setting minutes
 lcd_print(0,0,"Enter Minutes : ");
countdown_minutes = set_value(0 , 59);
 //  setting seconds
lcd_print(0,0,"Enter Seconds : ");
```

```
countdown_seconds = set_value(0 , 59);
```

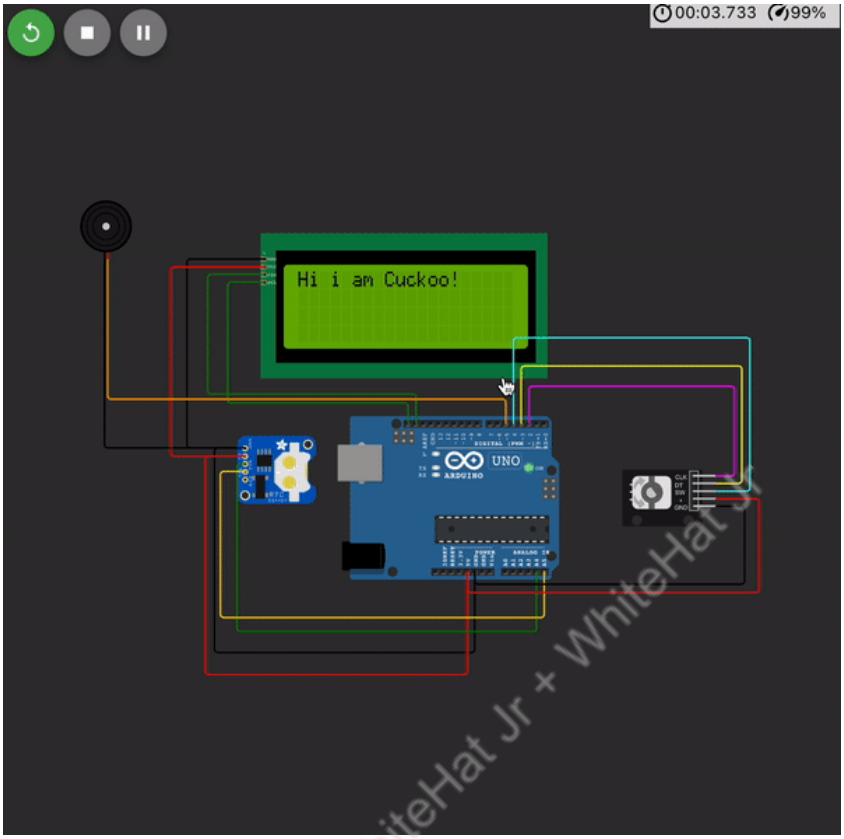
b. Until, either the minutes or seconds decrease to 0, we will continue to display the timer on the LCD. Even if seconds reach 0, we need to check the minutes and appropriately decrease it by a unit and update the seconds to 59.

c. Also, once it reaches 00:00, we will display **Countdown Over** on the display and play the buzzer to alert.

```
// start the stopwatch
while (countdown_minutes != 0 || countdown_seconds != 0){

 String t = String(countdown_minutes) + ":"
          + String(countdown_seconds);
 lcd.clear();
 lcd_print(0,0,"Countdown Timer");
 lcd_print(0,1,t);
 if (countdown_seconds == 0){
  if (countdown_minutes != 0){
   countdown_seconds = 59;
   countdown_minutes--;
  }
 }
 countdown_seconds--;
 delay(1000);
}
lcd.clear();
lcd_print(0,0,"Countdown over");
play_buzzer(200 , 3000);
```


Reference Output:



https://s3-whjr-curriculum-uploads.whjr.online/a73dbb57-a888-4a15-a9ad-d5dcd5ab0db9.gif

What's NEXT?

In the next class, we will learn to use another display hardware **LED Dot Matrix**.

Expand Your Knowledge

Read more about the LCD2004 here.