Over Temperature Alarm using a State Machine

Malcolm Knapp

Version 1.0

Workshop Weekend: Arduino

February 8th and 9th 2014

1. **Introduction:** We will start with a simple over temperature alarm and then extend it with more functionality. The code begins with a simple temperature set point above which the alarm goes off. Then we will add the ability to change the units of the temperature displayed. Finally we will add a warning condition that will tell you when you are close to going over the limit. This program uses a state machine which is a really powerful program structure. It can be used to construct complex behavior such as LCD menu systems, communication protocols, and robot controllers. So you can use a state machines for your own projects beyond this tutorial.
   - 1.1. For this tutorial we will use a button and a thermistor as input and an LCD as an output.
   - 1.2. This tutorial assumes that you know how to do the following. If you do not please do those tutorials first before continuing with this one.
     - 1.2.1. Add a library to Arduino
     - 1.2.2. Hook up an LCD to the Uno.
     - 1.2.3. Hook up a button to the Uno
     - 1.2.4. A SWITCH statement
   - 1.3. All code is called out in **bold,** all lines that you need to change are called out in **red bold** and lines you need to add are in **green bold.** Comments are marked with "[" "]"
   - 1.4. At various points you will have to insert code. The insertion points are marked with a with a number in a comment (e.g //-1-).

2. **Getting Started**
   - 2.1. Download the Over Temperature Alarm Tutorial 1V0 zip file from the Workshop Weekend: Arduino website.
   - 2.2. Unzip it. Within that folder you should see
     - 2.2.1. The Over Temperature Tutorial 1V0.pdf. This is the tutorial document that you are reading right now.
     - 2.2.2. The Over Temperature Alarm Start code. The is the code we will be modifying.
     - 2.2.3. Over Temperature Alarm Wiring Diagram. This is a Fritzing Diagram that shows you how to hook up the circuit.
     - 2.2.4. Thermistor_Example.ino. This is example is for testing to make sure the thermistor is hooked up correctly

3. **Hook up the circuit.** See the Over Temperature Alarm Wiring Diagram for details.
   - 3.1. Hook up the LCD
   - 3.2. Hook up the button to pin 6
   - 3.3. Hook up the thermistor to pin A0

4. **Add the needed libraries.** This tutorial uses two libraries
   4.1. The first is the Liquid Crystal Library which drives the LCD. It is built into Arduino so you do not need to do anything. Yay!
      4.1.1. The second is the Button Library. It makes reading the button easier. You can download it from the Workshop Weekend: Arduino website.

5. **Test the circuits**. I always use the example code provided with the library. That way you have known working code testing your circuit.
   5.1. To test the button
      5.1.1. Open File --> Examples --> Button --> Examples --> UniquePress
      5.1.2. To match the way we hooked up the button we need to change following line from:

      Button button = Button(12, BUTTON_PULLUP_INTERNAL);

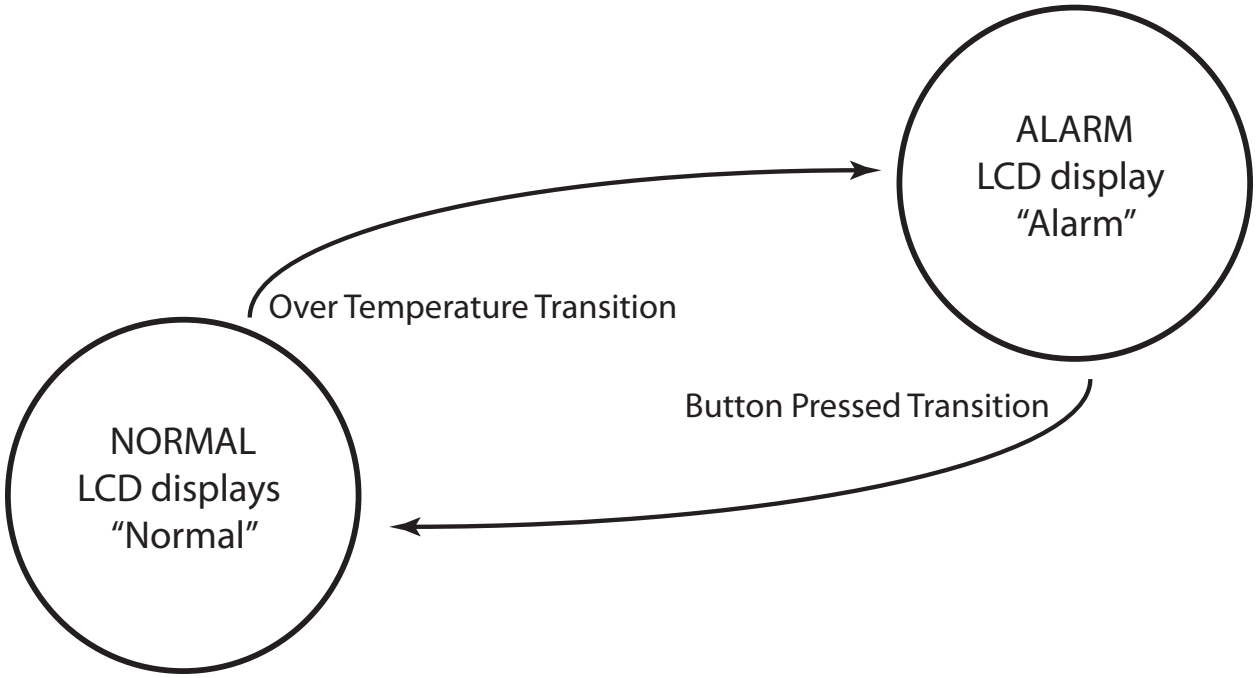
      to:

      Button button = Button(6, BUTTON_PULLUP_INTERNAL);

      5.1.3. Upload the code
      5.1.4. Click on the Serial Monitor in the upper right hand corner
      5.1.5. In the window that appears confirm that the baud rate drop down menu in the lower right hand corner is set to 9600. If it shows something different select 9600 from the menu.
      5.1.6. Confirm that you get a message every time the button is pressed. If you do not see this try and debug it your self and if that does not work ask for help.
   5.2. To test the LCD
      5.2.1. Open File --> Examples --> Liquid Crystal --> Hello World
      5.2.2. Upload Code
      5.2.3. Adjust Contrast potentiometer so you can see the text
      5.2.4. Confirm you see “Hello World” in line 1 and a counting second timer in the second line
   5.3. To test the Thermistor open the Thermistor_Example.ino from the Over Temperature Alarm folder
      5.3.1. Load the code
      5.3.2. Click on the Serial Monitor in the upper right hand corner of the Arduino IDE
      5.3.3. In the window that appears confirm that the baud rate in the lower right hand corner is set to 9600.
      5.3.4. Confirm that the temperature shown is between 50°F and 80°F and changes when you touch the thermistor.

6. **Our first State Machine**
   6.1. Open Over Temperature Alarm Start.ino.
   6.2. Save it with your own name.

6.3. Upload the code.
6.4. The state machine for this code looks like this:



6.5. Let's see how the transitions work for this state machine. Touch the thermistor with your finger. You should see the temperature on the LCD start to go up.
6.6. When the temperature goes above the OVER_TEMP_LIMIT of 80°F the system will transition from the NORMAL state and follow the arrow to the ALARM state and the LCD will start displaying "Alarm".
6.7. Now wait for the temperature to drop back below 80°F and you will see the state does not change. As you will soon see this is because there is no Event Capture in the ALARM state for the temperature going below 80°F
6.8. Press the button. This resets the alarm and the we transition back along the arrow to the NORMAL state.

7. **Code details**. State machines map to code extremely easily. It is one of the reasons why they are so powerful. We will go through the code piece by piece so you can understand it and then we will begin to extend it.
   7.1. Below the states of the Over Temperature Alarm have been isolated from the rest of the code to make it is easier to understand. All states are mapped to cases in a SWITCH statement. In this case the NORMAL state maps to the NORMAL case and the ALARM state maps to the ALARM case. The **break;** statement marks the end of each case. The current state is stored in the **state** variable. Each time the code goes around the loop the switch statement checks what the state is equal to and then jumps to that case.

```
switch (state) {
        case NORMAL: // NORMAL State
```

```
            [Entry and Event Capture Code]

        break;

        case ALARM: // ALARM State

            [Entry and Event Capture Code]

        break;
    }
```

7.2. The transitions arrows you see in the diagram are examples of events that can happen to the system. When an event happens it must be “registered” and then “captured” by the code. We will see how the code “registers” an event in a moment but let’s first look at how events are “captured”

7.3. All Event Captures are mapped to IF statements within each of case of the SWITCH statement. Below the Event Capture code has been isolated from each state. For the NORMAL state only the over temperature event is captured and for the ALARM state the only the button press is captured

```
// Event Capture
if (event == over_temp) {  // over temperature transition
        [Event Action Code]
        event = NONE;
}

----------------------

// Event Capture
if (event == button_pressed) {  // button pressed transition
        [Event Action Code]
        event = NONE;
}
```

At the end of every Event Capture the the event is set back to NONE. This means that that event code only is executed once per event which creates predictable behavior.

7.4. Within each Event Capture there is the Event Action code. This Action can be anything like turning on a motor or sending a message. In this case, the Event Action, in the over temperature Event Capture, is to change the state. In the button pressed Event Capture the Event Action is to display a “Ending Alarm” for 1 second and then change the state. The Event Action code has been isolated below

```
Serial.println ("over_temp event!");
state = ALARM;

----------------------

Serial.println ("end alarm");
lcd.clear();
lcd.print ("Ending alarm");
delay (1000);
state = NORMAL;
```

The powerful aspect of this is that Event Capture code is ***exclusive*** to the states that they appears in. This means you can overload a single button with a different behavior for each state the state machine has. You will see how to do this in a moment.

7.5. The other logic that is inside the each CASE statement is the state Entry code. This code is only executed when the state is changed.

```
// Normal State Entry Code
if (state != prevState) {
        [Entry Action Code]
        prevState = state;
}

----------------

//Alarm State Entry Code
if (state != prevState) {
        [Entry Action Code]
        prevState = state;
}
```

7.6. Within the State Entry code is the Entry Action code. It does all the things you want to do when you enter state for the first time. In this case all it does is change what is displayed on the LCD.

```
[State Entry Code]
Serial.println ("In NORMAL state, display Normal");
lcd.clear();
lcd.print ("Normal State");

----------------

[State Entry Code]
Serial.println ("in ALARM state, display Alarm");
```

```
        lcd.clear();
        lcd.print("Alarm State");
```

7.7. The last major piece of code associated with the state machine is the Event Polling code. This is the code that “registers” if an event as happened.

```
if (tempF > OVER_TEMP_LIMIT) {
        event = over_temp;
}

----------------

if (button.uniquePress()) {  // poll button
        Serial.println("button pressed!");
        event = button_pressed;
}
```

7.8. The event is then “captured” in the Event Capture code which you have already seen. This code is separated from the Event Capture code because it allows you to have the same Event Capture to be in multiple states without duplicating code.

7.9. One thing to note is that there is a precedence to the events i.e. that if two events happen at the same time, the event that is polled last in the code will be the one that is captured. It is possible to deal with this problem with using a queue data structure but that is beyond the scope of this tutorial.

7.10. The last part of the code is the base timer.

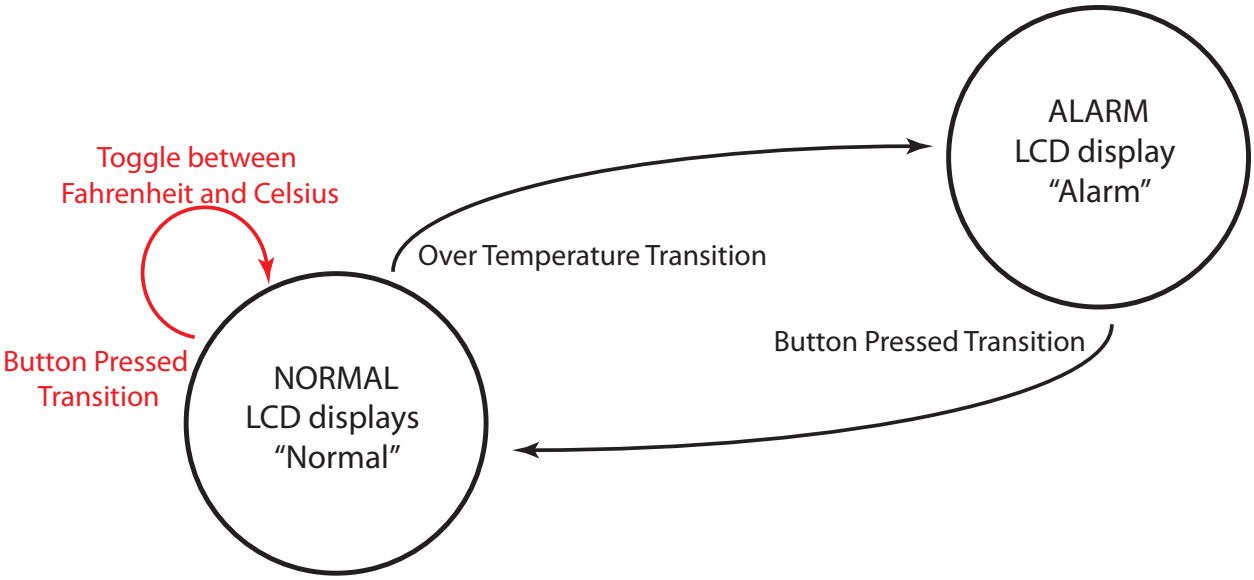
```
if(millis() - baseTimedOut > BASE_TIMEOUT) {
        baseTimedOut = millis();

        // update temperature display
        lcd.setCursor(0,1);
        lcd.print ("Temp is:");lcd.print (tempF); lcd.print("F");
}
```

This code is a non-blocking timer. It updates the temperature display every second

8. Now that you have seen the parts of the Over Temperature Alarm, let’s move on and show you how to extend it. The first thing we will do is to make it possible to display the temperature in either Celsius or Fahrenheit.

8.1. To do this we need to add an event to the NORMAL state where if the button is pressed the display will toggle between displaying Fahrenheit and Celsius units. This is an example of an event that does not cause a state change.



8.2. To begin with we need to add the Event Capture code. So within the NORMAL case code add the following code below the “-1-” tag

```
//-1-
if (event == button_pressed) {
        if (displayCelcius) {
                displayCelcius = false;
        } else {
                displayCelcius = true;
        }
        event = NONE;
}
```

8.3. The code will now capture the button_pressed event and set a variable that will determine which unit is displayed.

8.4. Next let’s add the code that uses that variable to determine what to display. So surround the line

```
lcd.print ("Temp is:");lcd.print (tempF); lcd.print("F");
```

with

```
if (displayCelcius) {
        lcd.print ("Temp is:");lcd.print (tempC); lcd.print("C");
        } else {
        lcd.print ("Temp is:");lcd.print (tempF); lcd.print("F");
}
```

8.5. Finally we need to initialize the variable so add the line below the -2- tag

```
//-2-
boolean displayCelcius = false;
```

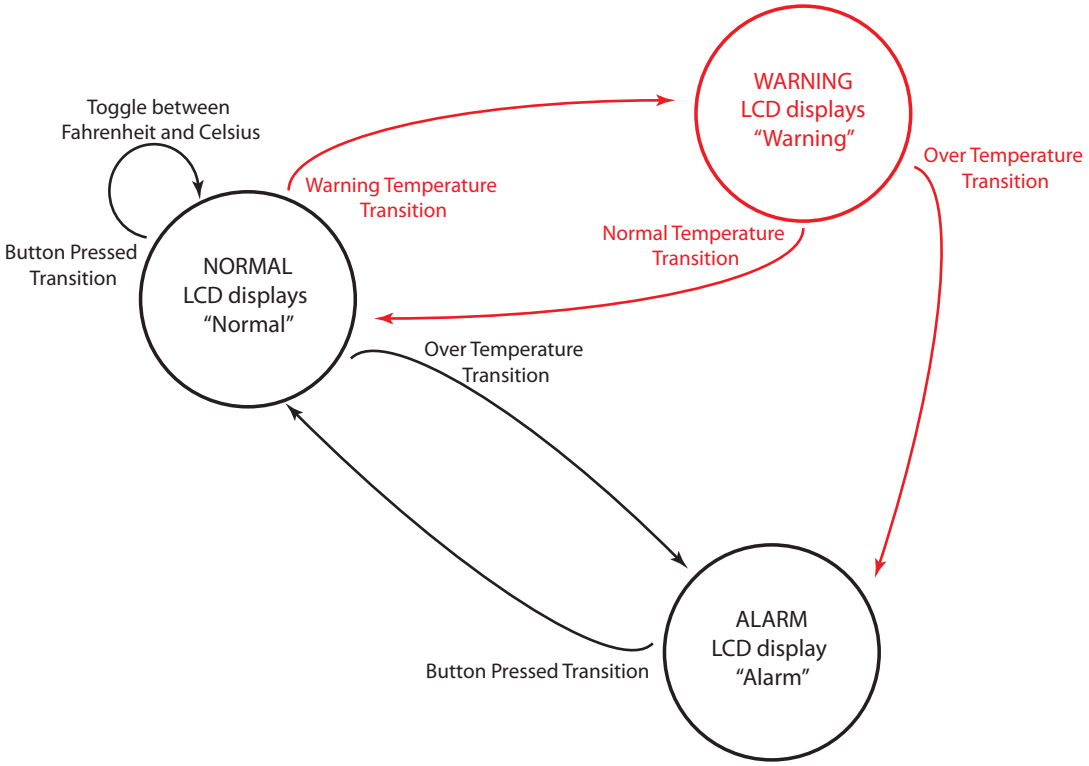
To the variable initialization section of the code.

8.6. Compile the code and make sure there are no errors.

8.7. Load the code. Now while you are in the NORMAL state each time you press the button the temperature display will switch between Fahrenheit and Celsius. However, if you are in the ALARM state the button will still end the alarm. The state machine has allowed you to overload the button with two functions. This is one of the reasons why it is such a powerful structure.

8.8. If you it does not work this way try and debug it on your own and then if it still does not work ask for help.

9. To complete the Over Temperature Alarm we will add a new state to the state machine This will be a WARNING state that displays a warning to the user if the temperature gets close to going over temperature but is not quite there yet. The state machine will look like this.



All the parts in red are the code we have to add. So we have to add one state and three event captures. The origin of the event transition line shows us where we need to put each of the event captures so we will need to add one event capture to the NORMAL state and the other two will be in the WARNING state. We will also have to add some variables that set what the Normal and Warning temperatures are.

9.1. To begin with let's create the WARNING state. Insert the code, at point -3-, between the NORMAL and ALARM states since it fits in there logically.

```
//-3-
case WARNING:

        if (state != prevState) {
                Serial.println ("in WARNING state, displaying Warning");
                lcd.clear();
                lcd.print("WARNING");
                prevState = state;
        }

break;
```

9.2. This only has the Entry code so far, showing what will be displayed on the LCD while the code is in the state.

9.3. Then let's add in the event capture code to the WARNING state. The normal_temp event captures the Normal Temperature Transition in the diagram. The over_temp event we have already seen.

```
case WARNING:

        if (state != prevState) {
                Serial.println ("in WARNING state, displaying Warning");
                lcd.clear();
                lcd.print("WARNING");
                prevState = state;
        }

        if (event == normal_temp) {
                Serial.println ("go to normal");
                state = NORMAL;
                event = NONE;
        }

        if (event == over_temp) {
                Serial.println ("go to alarm");
                state = ALARM;
                event = NONE;
        }

break;
```

9.4. Then we need to add the warn_temp event capture to the NORMAL state. Insert the following code at point -4-.

```
//-4-
if (event == warn_temp) {
        Serial.println ("warn_temp event!");
        state = WARNING;
        event = NONE;
}
```

9.5. Then we need to update the Event Polling section. So add the following code above the OVER_TEMP_LIMIT check at point -5-.

```
//-5-
if (NORMAL_TEMP_LIMIT > tempF ) {
        event = normal_temp;
}

if (tempF >= WARN_TEMP_LIMIT) {
        event = warn_temp;
}
```

9.6. Finally we need to initialize the variables that store all this information. First add the NORMAL_TEMP_LIMIT and the WARN_TEMP_LIMIT variables at point -6-.

```
//-6-
double NORMAL_TEMP_LIMIT = 72;
double WARN_TEMP_LIMIT = 76;
```

9.7. Then update the state list with the WARNING state. Be sure there is a comma after it.

```
enum {
        // States
        NORMAL = 1,
        WARNING,
        ALARM,
        NO_STATE
} STATES;
```

9.8. Lastly update the event list with normal_temp and warn_temp. Be sure to add the comma after it.

```
enum {
        button_pressed = 0,
        normal_temp,
        warn_temp,
        over_temp,
        ENTRY,
```

NONE
} EVENTS;

9.9. Compile the code and make sure there are no errors

9.10. Load the code. What will now happen is that as the temperature goes up you will first enter the WARNING state. If you then let go of the thermistor the temperature will drop and you will go back to the NORMAL state. But if you keep holding on to the thermistor the temperature will go over the OVER_TEMP_LIMIT and the code will go to the ALARM state. Then it will take a button press to go back to the NORMAL state.

9.11. If you it does not work this way try and debug it on your own and then if it still does not work ask for help.

9.12. If everything works, you are DONE! You now have a fully working Over Temperature Alarm.

10. If you have completed the tutorial up to this point here are a few things to try, to play a bit more with state machines.

10.1. Add a second button to the circuit and change the code so that when you are in the NORMAL state, one button increases the OVER_TEMP_LIMIT and the other decreases it.

10.2. Add an ALARM_ACKNOWLEDGE state to the the state machine. Follow the diagram below.