Mech 368 Lab 2

Thomas Richmond, 52228848

08 October 2021

# Exercise 1

## Goals

* Compute the required resistances for LEDs
* Create a simple LED circuit

## Setup

Resistor value computation for input voltage and current

|  |  |  |  |
| --- | --- | --- | --- |
| **LED Color** | **VDrop**  **[V]** | **R, Ideal**  **[**Ω**]** | **R, Standard**  **[**Ω**]** |
| Red | 2,1 | 290 | 470 |
| Yellow | 2,1 | 290 | 470 |
| Green | 3,4 | 160 | 220 |
| Blue | 3,4 | 160 | 220 |

Table

Description automatically generated

# Exercise 2

## 2.1 Goals

* Create a simple switch
* Identify and resolve bouncing using a debounce circuit

## 2.2 Setup

The following board setup shows the setup for a bouncy switch circuit (right) and a debounced switch circuit. (left) The orange wires are scope signal inputs.

A picture containing diagram

Description automatically generated

A 10kΩ resistor was selected as per the lab instructions. A 10µF capacitor was selected according to lab guidelines of selecting a capacitor between 1nF and 100nF.

## 2.3 Data

### Bouncy Circuit

A trace of the output voltage from the switch is shown below. The bounce is clearly seen as the switch is opened, with two very clear peaks in the image below.

Diagram

Description automatically generatedDiagram

Description automatically generated

### Debounced Circuit

The debounced circuit is able to prevent the signal from rising to 5V. Some bounce is still observed, as seen below.

Diagram

Description automatically generated

Looking specifically at the rising curve as the switch opens, we can calculate the time constant for the circuit using the standard formula

|  |  |
| --- | --- |
| By the data measured by the oscilloscope,  Comparing this with the value expected nominal value  Assuming no innate error in resistance or capacitance values, the error for this measurement is | Chart, line chart  Description automatically generated |

# Exercise 3

## Goals

* Interface with the EXP board LEDs using C#
* Compare signal traces obtained from the AD2 oscilloscope

## Setup

A .NET Windows Forms App is used to drive the EXP board LED. The code used is as follows:

|  |  |
| --- | --- |
| using System;  using System.Drawing;  using System.Windows.Forms;  using System.IO.Ports;  using System.Threading.Tasks;  namespace Mech368L2E3  {  public partial class Mech368L2E3: Form  {  enum AD2Token  {  LED\_ON = '5',  LED\_OFF = '%'  }  const int MAX\_LED\_INTERVAL = 3000;  const int MIN\_LED\_INTERVAL = 500;  readonly Color CON = Color.DarkBlue;  readonly Color COFF = Color.DarkGray;  SerialPort port = new SerialPort();  public Mech368L2E1()  {  InitializeComponent();  InitializeSerial();  Random generator = new Random();  ledTimer.Interval =  generator.Next(MAX\_LED\_INTERVAL - MIN\_LED\_INTERVAL)  + MIN\_LED\_INTERVAL + 1;  ledTimer.Tick += this.LEDOn;  ledTimer.Tick += (sender, e) =>  Task.Delay(MIN\_LED\_INTERVAL).ContinueWith(  t => this.LEDOff());  ledTimer.Enabled = true;  }  private void InitializeSerial()  {  this.port = new SerialPort("COM3");  this.port.Open();  }  private void LEDOn(object sender, EventArgs e)  {  // Write to EXP board  this.port.Write(((char)AD2Token.LED\_ON).ToString());  // Display on UI  this.ledButton.BackColor = CON;  }  private void LEDOff()  {  // Write to EXP board  this.port.Write(((char)AD2Token.LED\_OFF).ToString());  // Display on UI  this.ledButton.BackColor = COFF;  }  private void OnFormClosing(object sender, EventArgs e)  {  if (this.port != null && this.port.IsOpen)  this.port.Close();  }  }  } | **Comments:**  *Enumerate the EXP board commands*  *Timer tick rate is randomly generated within the range (MIN\_LED\_INTERVAL, MAX\_LED\_INTERVAL]*  *Timer ticks trigger two events:*   1. *Turn LED on* 2. *Set up timer to turn LED off after a 500ms delay*   *LED is also virtually displayed on a form* |

## Data

From the oscilloscope we see the following trace, where the switch signal and LED signal are connected to channel 1 (yellow line) and channel 2, (blue line) respectively.

A picture containing graphical user interface

Description automatically generated

Timeline, treemap chart

Description automatically generated Zooming in on the edges, and performing some measurements, we find that I have a reaction time of about 191.7ms.

# Exercise 4

## Goals

* Implement and use a thermistor
* Test thermistor in real-life scenarios

## Setup

The thermistor circuit is set up according to the lab guidelines with a resistor R = 10kΩ.   
From the β25/85 =3435K thermistor datasheet,

* + From this, ,

Table

Description automatically generated

## Data

We can use the following approximation for temperature:

An input voltage of 5,0V was used below.

*Continued…*

### Ambient Conditions

In ambient conditions, , corresponding to .

*(My thermometer says 18C)*

A picture containing graphical user interface

Description automatically generated

In an ice bath at , , corresponding to .

Graphical user interface

Description automatically generated

In my lukewarm cup of tea, , corresponding to .

*(Unfortunately, I could not find my thermometer, so I do not have a value for temperature of the tea.)*

Graphical user interface

Description automatically generated

There is a significant error in the ice water (many degrees over zero measured!) and this can perhaps be resolved by fitting a better curve to the data, or by introducing an offset. If a reliable reference is used, we can plot response curves (say, between immersing the thermistor from ice water to hot water) and fit the thermistor to the reference.

More interesting is the error introduced by internal heat generation – that is, some heat is generated by virtue of current moving across the thermistor - which may affect the accuracy of the readout, even if the above compensation is implemented. To manage this self-heating effect, consider that the power dissipated in the thermistor is

The thermistor will radiate heat to the environment, roughly according to the relation

If we equate the heat generation and radiation, we can determine yields

V If we can fix the value of *h*, then we can relate the self-heating-induced temperature increase of the thermistor to the power being injected into the resistor, recalling that V and R are measurable values. In this case, *h* has SI dimensions of .

# Exercise 5

## Goals

* Use the waveform function of the AD2
* Explore high-pass and low-pass RC circuits
* Compute time constant using an oscilloscope

## Setup

The following generic circuit setup was used:

A picture containing calendar

Description automatically generated

The output signal is obtained from the orange wire. For the high pass circuit, the blue wire is connected to the waveform and the cyan wire is grounded. The setup is vice-versa for a low-pass circuit.

Time constant is computed by finding where the output voltage has decayed by 63% - that is, when the elapsed time is equal to the time constant. For a five-volt signal, this corresponds to

## Data

The following data was obtained using an oscilloscope. **Refer to Appendix A for images of the data acquisition** code.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R** | **C** | **Expected** | **High-Pass Time Constant** | **Error** |
| 1k | 100nF | 100us | 98.66 | 0,0135 |
| 10k | 100pF | 1us | 1.234 | 0,1896 |
| 10k | 100nF | 1ms | 1,02 | 0,0196 |
| 10k | 1nF | 10us | 10,22 | 0,0215 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **R** | **C** | **Expected** | **Low-Pass Time Constant** | **Error** |
| 1k | 100nF | 100us | 99.68 | 0,0032 |
| 10k | 100pF | 1us | 2,78 | 0.6403 |
| 10k | 100nF | 1ms | 1,02 | 0.0196 |
| 10k | 1nF | 10us | 16,7 | 0.4012 |

The error recorded generally agrees with the uncertainty of the components. Some larger error may be due to the error in reading from the waveform graph given that the circuit response gets very quick.

# Exercise 6

## Goals

* Measure the response of a thermistor submerged in ice water
* Apply the decay equation to a temperature system

## Setup

Use the thermistor circuit from Exercise 4 for this lab. The temperature time constant corresponds to a (absolute, i.e. Kelvin) reduction in temperature of . We can use this time constant for the standard decay formula applied over temperature,

A Windows Forms App was developed for real-time graphing and datalogging. Refer to Appendix B for the code.

## Data

The thermistor is submerged into ice water measured to be 0,3 degrees Celsius. The response data was logged and is plotted below:

The thermistor is submerged at 5.23s elapsed time. We see the signal decays from 25,6 to 11,7 degrees Celsius. (The error in this measurement, though very significant, is irrelevant as the time constant should be independent of temperature difference) A decay of 63,2% corresponds to a temperature of 16.8 degrees Celsius. This occurs at 14.1s of elapsed time. The thermal time constant for the system is:

# Appendix A

|  |  |  |  |
| --- | --- | --- | --- |
| **R** | **C** | **High-Pass Trace** | **Low-Pass Trace** |
| 10k | 100pF | Diagram  Description automatically generated | Diagram  Description automatically generated with low confidence |
| 10k | 100nF | **\*** | A picture containing graphical user interface  Description automatically generated |
| 10k | 1nF |  |  |
| 10k | 100pF |  |  |

\* The high pass 10k/100pF circuit was unable to reach 5V. The same 63% decay rule applies as described in the Exercise 5 text, but for a lower voltage.

# Appendix B

|  |  |
| --- | --- |
| using System;  using System.IO;  using System.IO.Ports;  using System.Windows.Forms;  using System.Collections.Generic;  using System.Collections.Concurrent;  using System.Windows.Forms.DataVisualization.Charting;  namespace Mech368Lab2E6  {  public partial class Mech368L2E6 : Form  {  enum AD2Token  {  PACKET\_HEADER = 0xFF,  STREAM\_REQUEST\_ACCELEROMETER = 'A',  STREAM\_REQUEST\_THERMISTOR = 'c',  LED5\_ON = '5',  LED5\_OFF = '%'  }  private ConcurrentQueue<int> streamBuffer;  private SerialPort port;  private StreamWriter streamWriter;  private Series temperatures;  public Mech368L2E6()  {  InitializeComponent();  InitializeListeners();  this.streamBuffer = new ConcurrentQueue<int>();  this.port = this.BindSerial();  this.port.Write(  ((char)AD2Token.STREAM\_REQUEST\_THERMISTOR)  .ToString());    this.temperatures = new Series("Temperatures");  this.temperatures.ChartType = SeriesChartType.FastLine;  this.temperatures.YValueType = ChartValueType.Double;  plot.Series.Add(this.temperatures);  }  private void InitializeListeners()  {  this.filepathTB.Click +=  this.ConfigureOutputStream;  this.savetofileCB.CheckedChanged +=  this.ToggleFilewriter;  this.processLoop.Tick +=  this.ProcessDatastream;  }  private SerialPort BindSerial()  {  SerialPort p = new SerialPort()  {  PortName = "COM3",  BaudRate = 9600,  Parity = Parity.None,  DataBits = 8,  StopBits = StopBits.One,  Handshake = Handshake.None  };  p.DataReceived += this.CaptureDatastream;  p.Open();    return p;  }  private void ConfigureOutputStream(  object sender,  EventArgs e  ) {  SaveFileDialog sfd = new SaveFileDialog();  if (sfd.ShowDialog() == DialogResult.OK)  filepathTB.Text = sfd.FileName;  }  private void ToggleFilewriter(object sender, EventArgs e)  {  if (this.streamWriter == null)  if (this.filepathTB.Text.Length > 0)  this.streamWriter = new StreamWriter(  this.filepathTB.Text);  else  MessageBox.Show("Invalid filename!");  else  {  this.streamWriter.Close();  this.streamWriter = null;  }  }  private void OnFormClosing(object sender, EventArgs e)  {  if (this.port != null && this.port.IsOpen)  this.port.Close();  if (this.streamWriter != null)  this.streamWriter.Close();  }  private void CaptureDatastream(object sender, EventArgs e)  {  int next; // Byte buffer  try  {  while ((next = this.port.ReadByte()) != -1)  this.streamBuffer.Enqueue(next);  }  catch { /\*Ignored!\*/ }  }  Queue<int> circularBuffer = new Queue<int>();  private void ProcessDatastream(object sender, EventArgs e)  {  void EnqueueCircular(  Queue<int> queue,  int next,  int max  ) {  queue.Enqueue(next);  if (queue.Count > max)  queue.Dequeue();  }  int AverageQueue(Queue<int> queue, int offset = 0)  {  float avg = 0;  foreach (int val in queue)  avg += val - offset;  avg /= (float)queue.Count;  return (int) Math.Round(avg);  }  int signal = -1;  while (this.streamBuffer.TryDequeue(out int next))  {  if (next == (int)AD2Token.PACKET\_HEADER)  {  if (signal == -1)  {  signal = 0;  continue;  }  int nextVoltage = signal \* (3300 / (2 << 9));  EnqueueCircular(  this.circularBuffer,  nextVoltage,  100);  int avgVoltage = AverageQueue(  this.circularBuffer);  voltageTB.Text = nextVoltage.ToString();  voltageAvgTB.Text = avgVoltage.ToString();  double Temperature(int millivoltage)  {  double B = 3435, T25 = 25 + 273; // K  double R1 = 10; // kOhms  double Rth = R1 \* millivoltage  / (3300 - millivoltage);  double T = 1/(1/B \* Math.Log(Rth/R1)  + 1/T25);  return T - 273;  }  double temperature = Temperature(avgVoltage);  tempTB.Text = temperature.ToString("0.#");  this.temperatures.Points.AddY(temperature);  if (this.streamWriter != null)  this.streamWriter.WriteLine(  $"{DateTime.Now.ToString(  "hh.mm.ss.ffffff")}, "  + $"{nextVoltage}, "  + $"{avgVoltage}, "  + temperature.ToString("0.##"));  signal = 0;  continue;  }  else if (signal == -1)  continue;  else // Assemble bytes  signal = (signal << 5) + next;  }  }  }  } | **Comments:**  Maintain a list of |

Graphical user interface, application

Description automatically generated