Converting the Multivibrator programs into a User Defined Function Block

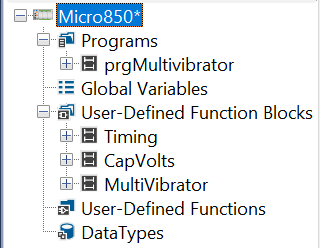
## By

## Klint Mahne

## Lessons Worth the Struggle

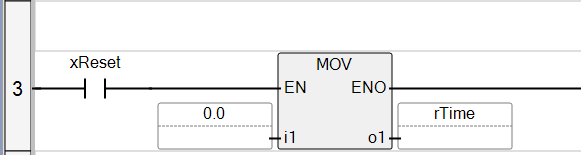
I must admit when I first undertook the task of creating the multivibrator, I figured it was going to be a math-ridden complication and expected a good deal of struggle as it was more than just making blinking lights go on and off at intervals – replicating the components would require functions and thoughtful placement for it to properly operate. This went about as expected – a few hiccups here and there but overall, nothing that was absolutely stumping. Having made some simple User Defined Function Blocks before, I anticipated this tutorial was going to be extremely easy and almost not even worth writing. I was dead wrong. The complications that arose from eliminating the global variables and instantiating blocks within blocks were a bit demoralizing. Here is the catch – after moving, troubleshooting, correcting, and completing the project I realized that the problem originated not from the code itself, but from my lackadaisical approach. Ladder Logic is a type of programming code, after all, and when you start moving things around without the proper care, it will brutally remind you of every little gap in understanding you might have with whatever you are working with. So, with all of that confidence I had going into this tutorial, I now walk out on the other side victorious, but humbled and with a much better understanding of the individual parts.

## Understanding the User Defined Function Block (UDFB)

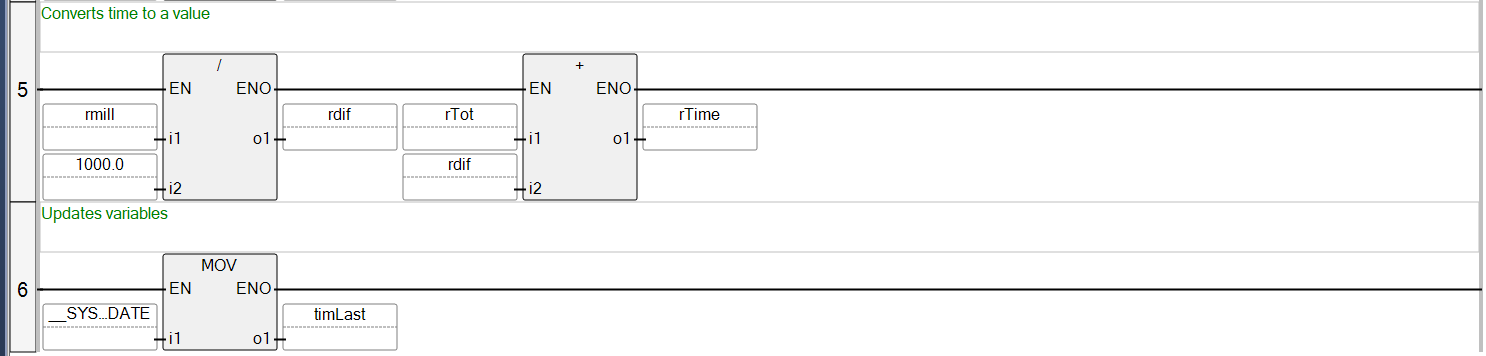
The previous tutorial saw the creation of a program to replicate the operation of a multivibrator in ladder logic. Although successful in both replicating the components in a way that allows the user to define voltage and capacitance to alter the timing, it also consisted of 3 separate programs running concurrently. The prgTime program captured the time elapsed between specific events required to emulate the time that would pass in an analog environment. The prgCapVolts program contained all the data and calculations that simulated the characteristics of a capacitor in its operation as part of the multivibrator. The prgCtrl program is where the prgTime and prgCapVolts global variables interacted with each other, coming together to create the actual multivibrator function itself. The prgOutMap existed solely to map the internal LED variables to a physical output of the PLC, which is a good idea to use whenever interfacing input/output components with the PLC.

This tutorial aims to make the prgTime program and prgCapVolts programs into their own UDFBs, and then consolidate them to a single overall UDFB which operates as the multi-vibrator. This in turns helps to eliminate the need for many global variables to make a fully functional, succinct, localized, easy to follow, and most important function block that can be used multiple times in the same program. Keep in mind, though, that a UDFB’s variables are usually loaded through inputs and outputs. When creating the local variables, it is important to designate what is a VariableInput, VariableOutput, and a standard Variable impacting only the UDFB itself.

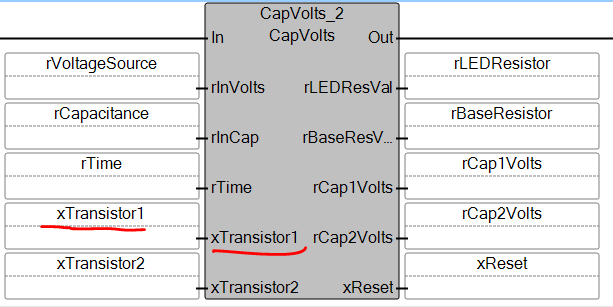
## Transforming the Timing Program

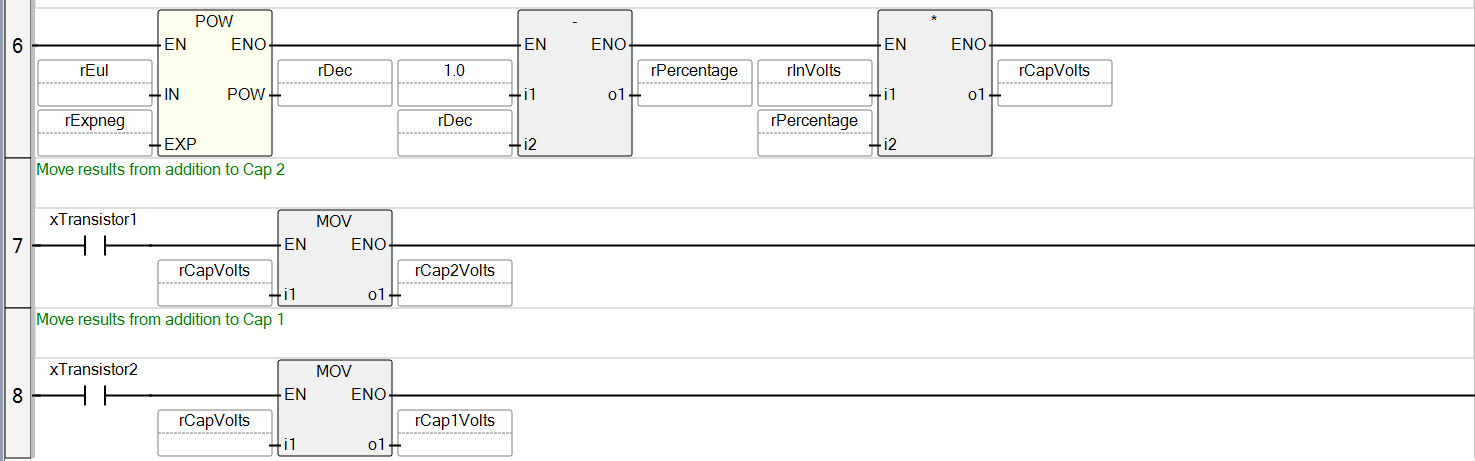
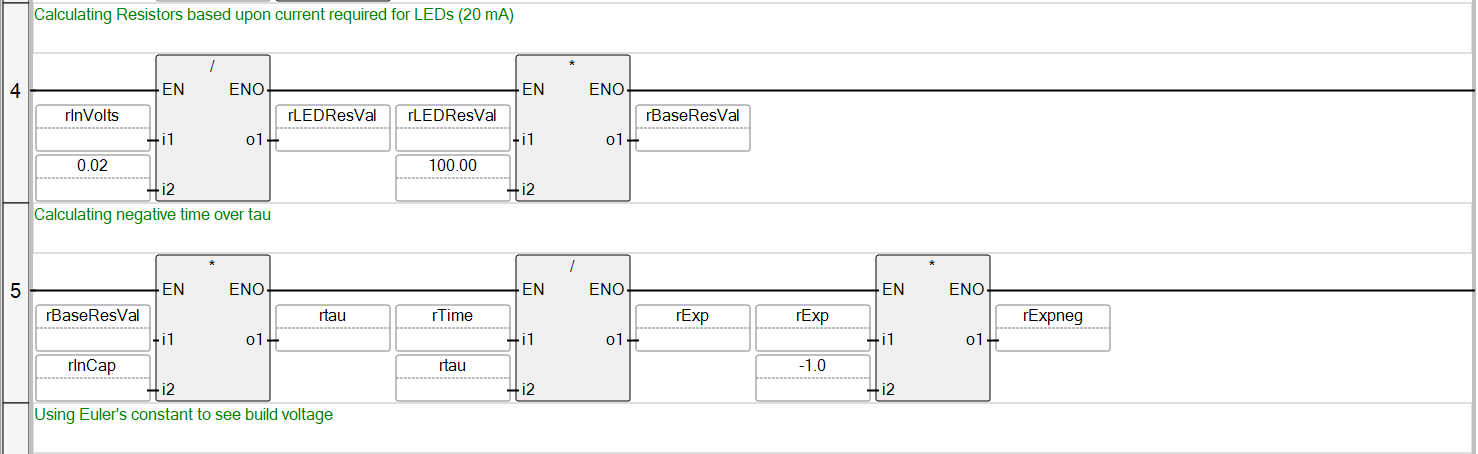
Functionally, the multivibrator’s timing calculations are going to be the same regardless of if it has its own UDFB or if it’s contained in a program. The main difference between the two is not going to be in the logic, but rather strictly in the variables. Remember – creating User Defined Function Blocks means that we want locally contained variables, not global variables. The timing program was, for the most part, directly convertible into a UDFB. The main challenge here was now that we were no longer using global variables, another block or program is not able to manipulate the variable inside of the timing UDFB. This required the ‘zeroing’ or reset of the time variable to move from the prgCapVolts program into the timing function itself. This was done by adding an input variable called ‘xReset’ that would zero out the time. Although this is technically a bit more work as it requires the functions to communicate to each other through inputs and outputs instead of communicating through global variables, it feels a bit more intuitive for the process to reside inside of the timing function block. The result is a timer that counts until it is reset, much like a stopwatch which can now be used in other programs as needed. Note that in the full picture you will notice the first line is an ‘In’ to and ‘Out’. This is to allow the UDFB to exist on a rung without blocking the enable line.





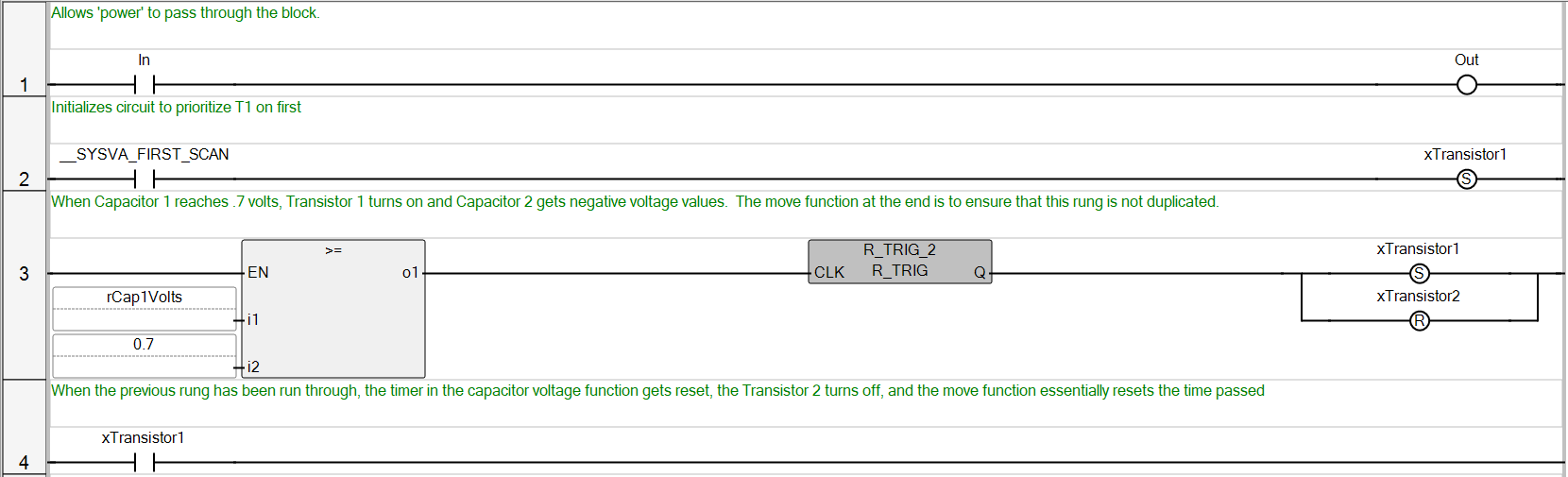
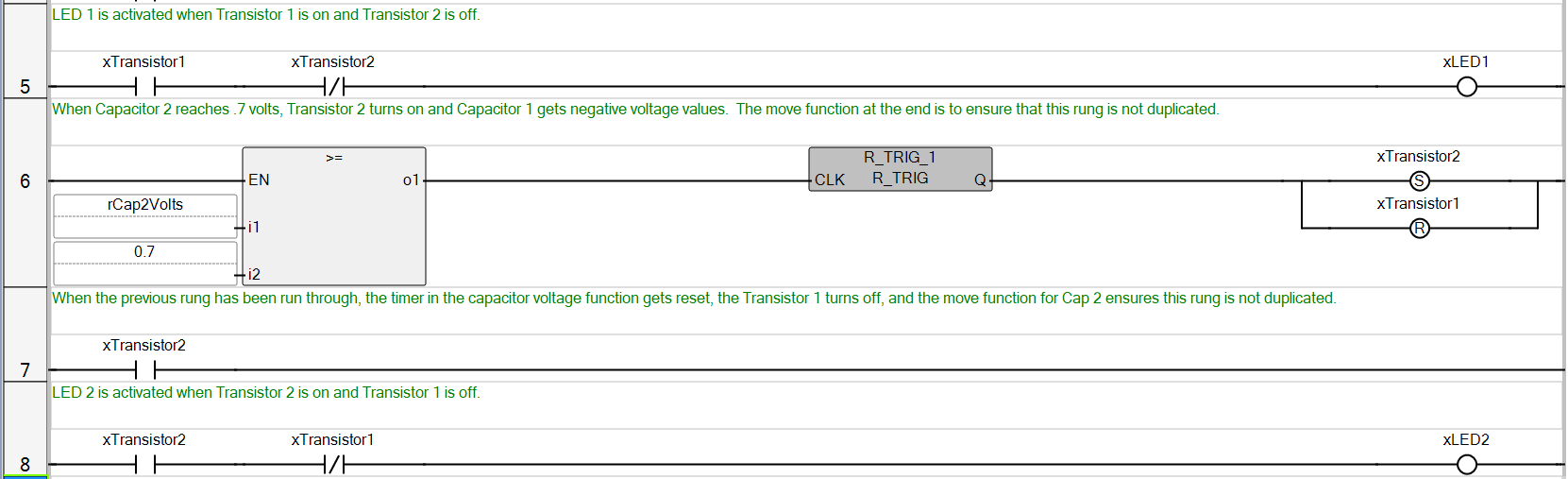
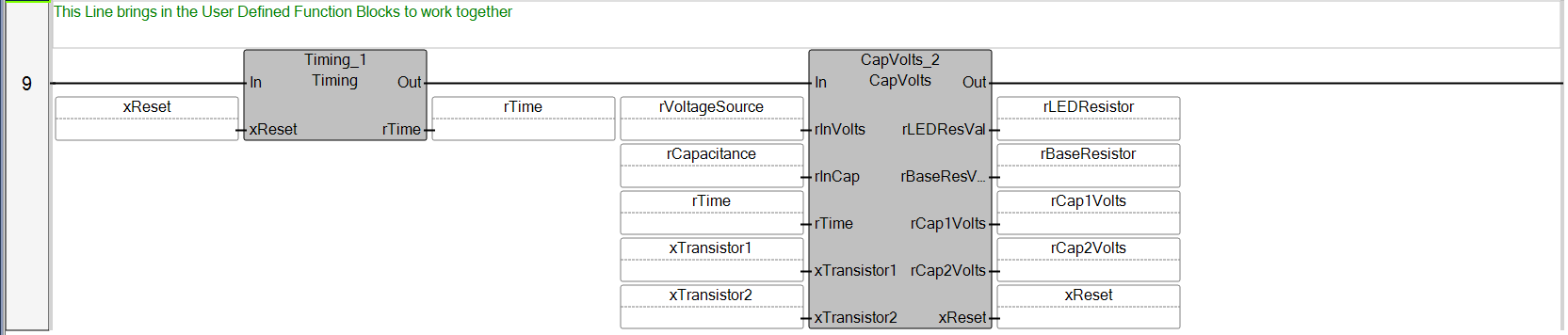
## Converting the Capacitor Program

I can confidently say that this program was personally the toughest to convert. It required me to think about the components and variables in ways that I did not really need to before. In the previous tutorial I made it change using math, but for this it would require me to understand what the change truly meant and how it impacted the other components. When I originally built the program, the calculations and manipulation of the capacitors was based off the relay expressions of the transistors. I had to do that same thing again here, but with the transistors existing outside of the variable scope a new challenge had arisen. Frustrated at how much more difficult this was than I anticipated, I wanted so badly to go back to using global variables. That is until I realized that I could make what amounts to a global variable substitute. The beauty of using local variables is that they can be named whatever they need to be, and it will not affect other UDFBs nor programs. In following the logic, I was able to use the same names for the transistor variables outside of the UDFB as the transistor variables inside of the UDFB by way of matching the inputs. The same could be done for the time variables. Just like that, the problem with not having access to global variables melted away. Of course, this practice will not always work – in some applications it may cause more confusion for someone needing to trouble shoot it. For this application, though, it works just fine. Once past this hurdle, the rest fell into place.

For most of the remaining code, it translated from the program into the UDFB easily enough. Because we had to move the function that reset the time from this block into the timing UDFB, there was some adjustment needed. You will notice that there is the xReset on the output; this is the same xReset that is found on the input of the timing UDFB. In the CapVolts UDFB it is reset on .8 rather than .7 volts. This hysteresis allows a bit of wiggle room in the event that the voltage source is high, ensuring that the .7 doesn’t get skipped over.

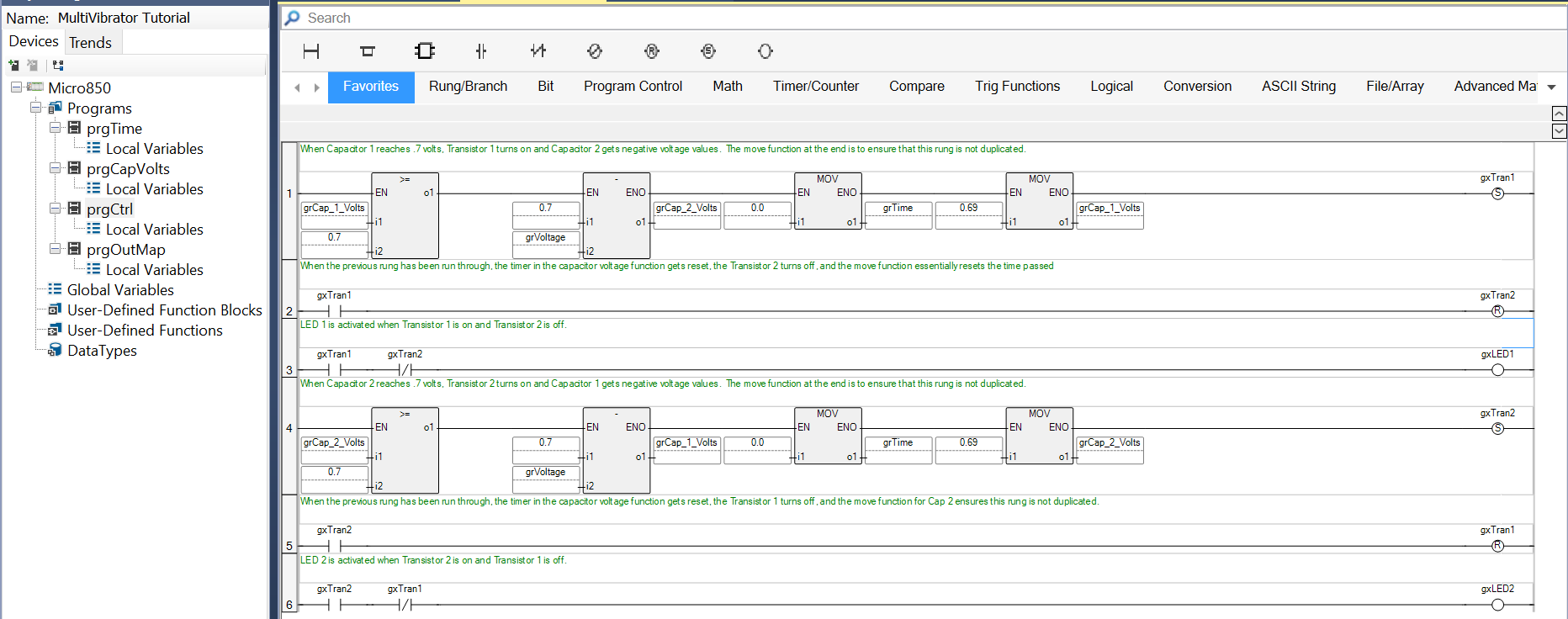
## Main Modification of the Multivibrator

Both the timing and capacitor voltage calculating UDFBs were instantiated to exist in this main UDFB. The goal, after all, was to have a single UDFB in the program. Because of this, the Multivibrator UDFB acted not just as the home for the LED and Transistor representations of the logic like the program was, but it served as a messenger between the two other UDFBs. This meant that the Multivibrator UDFB was not something that had gotten moved and slightly altered, but rather it evolved alongside and was built out simultaneously with the others. After all, if the capacitor voltage UDFB required the transistor representation to work for it to operate correctly, then it only makes sense that the layers containing the transistor representation would needed to have been built out and tested as well.

It’s not that there were no lessons learned at this point, or that everything happened smoothly, but rather the lesson was how intertwined everything might be, and how important it is to know what can be separated and what cannot. Remember, though, that the point of the multivibrator was not meant to just show the lighting of the LEDS back and forth, but also to provide resistor values for the given voltage and capacitance. This block, then, would need to have two inputs (voltage and capacitance) and 4 outputs (LED1, LED2, resistors on transistor base, and resistors over LEDs).

## The Main Program

I could explain and re-explain the benefits of User Defined Function Blocks and how you can build a library so they can be used repeatedly or how you can put multiple instances of the same function in the same program to compare the differences. I feel like the benefits can speak for themselves, though. So with that, I will leave you with this:   
  
Main Program Before UDFB:

  
  
  
Main Program After UDFB:

