Introduction:

MouseBot is a mouse-like robot, without being a pest. MouseBot has 4 Infrared sensors, used as proximity sensors, and will sense when something is close to it. When something is close to it, it will sense where that object is, and then will “run” away from the object.

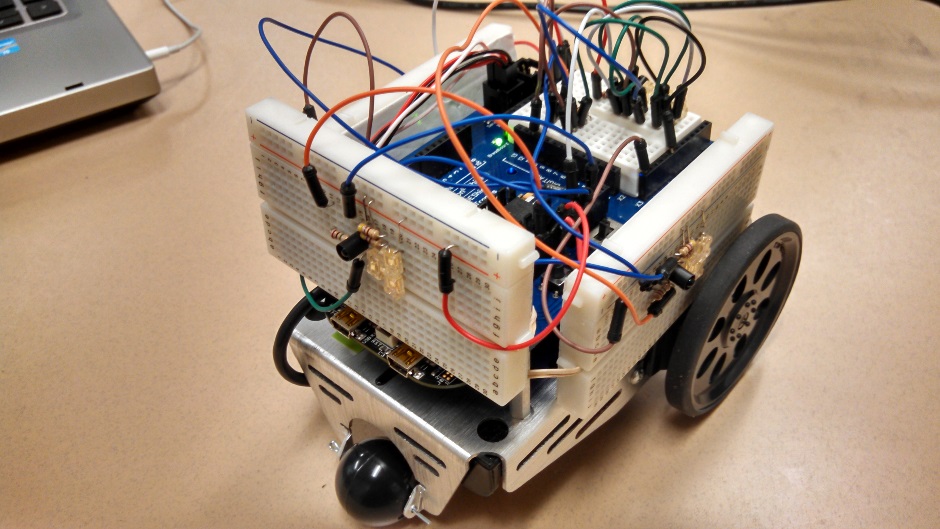
This is a very simple design to prove a very large perspective, Autonomous Robots. When that word comes to mind, I think of the Google car, but in reality, even MouseBot is an autonomous robot. An autonomous robot includes anything that handles a task completely by itself, and MouseBot’s only task is to run away from objects! How delightful!

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Description | Item | Amount | Price | Cost | Supplied? |
| IR Photodiode | OP245A | 4 | $0.74 | $2.96 |  |
| IR Emitter | LTR-4206E | 12 | $0.38 | $4.56 |  |
| Resistor | Misc. Values | 12 | $0.04 | $0.53 |  |
| Microcontroller | KL25Z | 1 | $12.95 | $12.95 | Yes |
| Robot Shield | 130-35000 | 1 | $134.99 | $134.99 | Yes |
| Misc. | Wires, Etc. | 1 | $10.00 | $10.00 | Yes |
| Total | | | | $165.99 |  |
| Total (minus supplied parts) | | | | $8.05 |  |

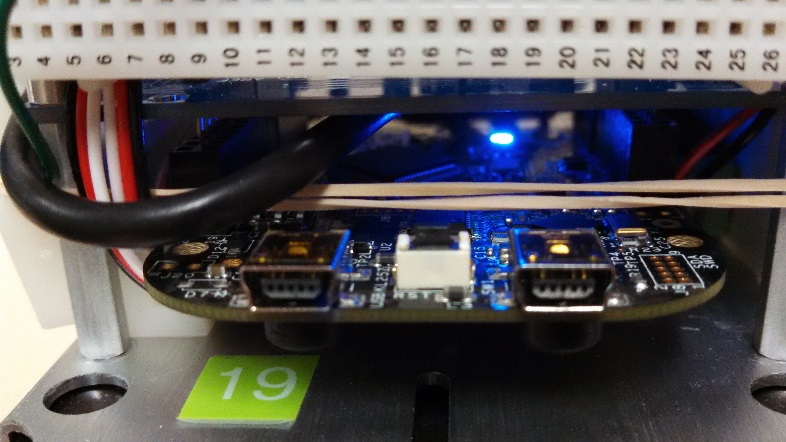
Bill of Materials, everything needed to build a MouseBot.

Design:

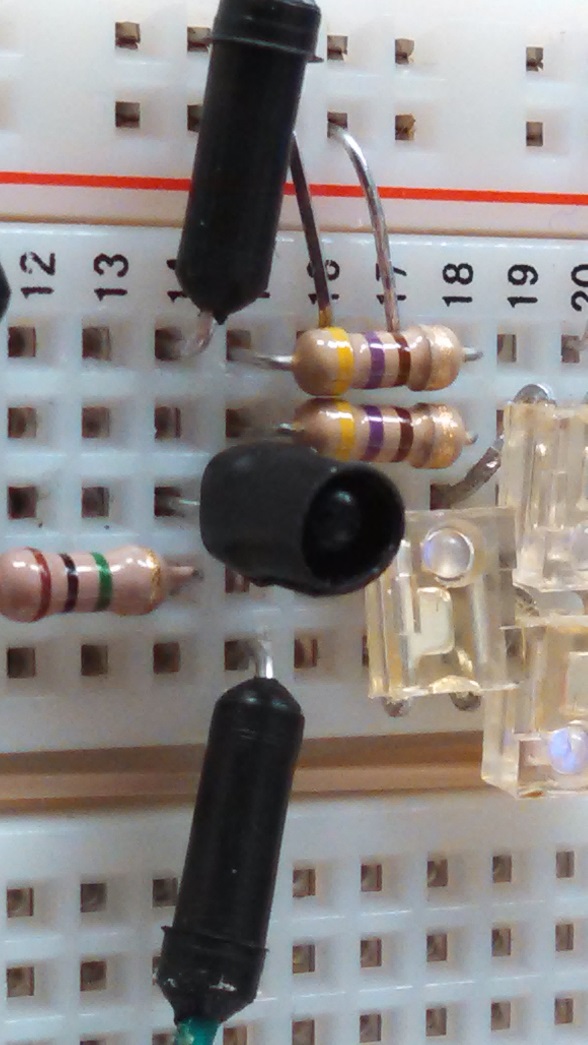
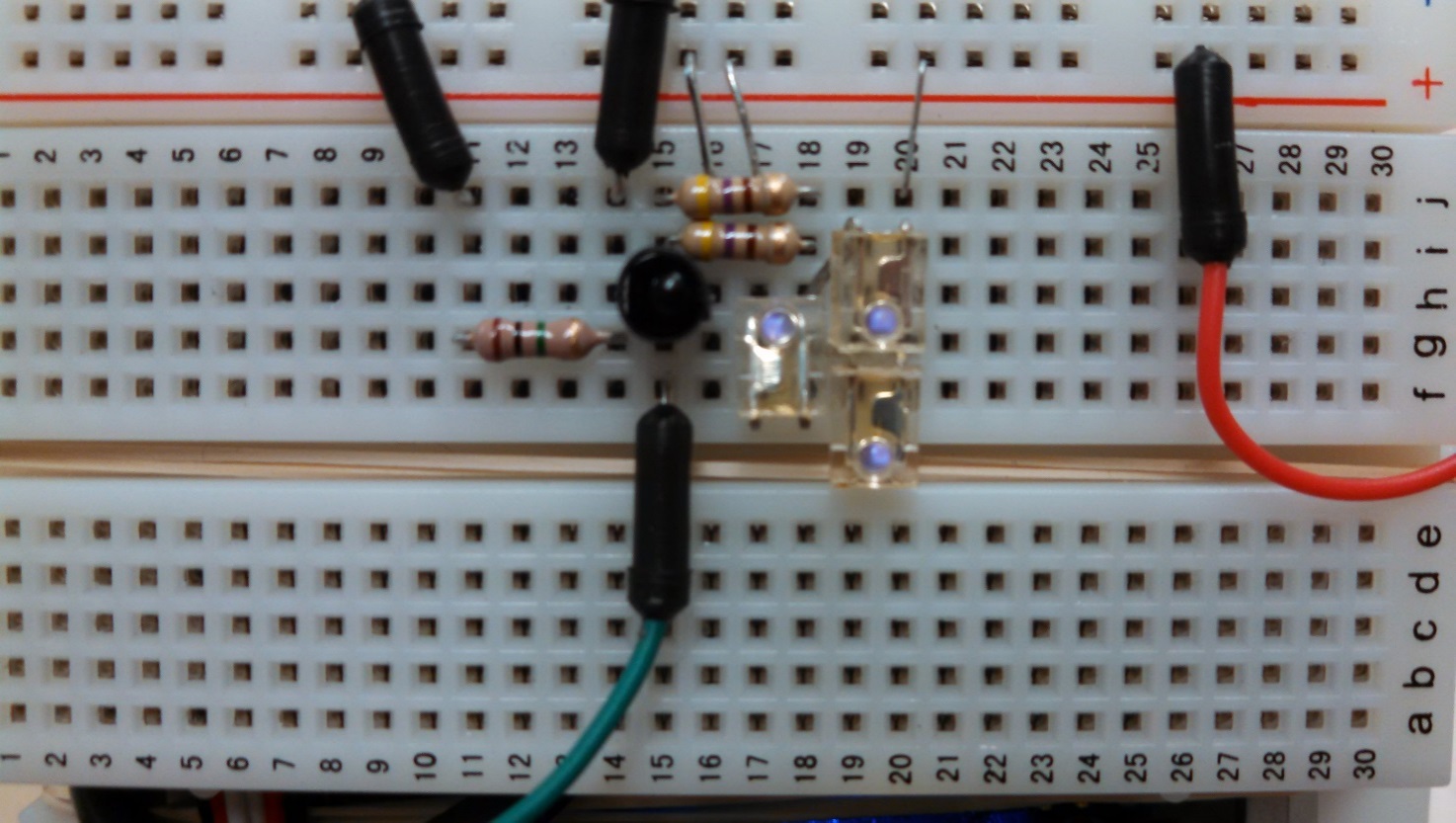
The Design of MouseBot wasn’t that hard to plan, especially since we had a robot shield that included a frame, two servos, and a battery pack for mobile performance. Knowing that the shield is made for Arduino, I chose to use the FRDM KL25Z128 Microcontroller, because who wants to simply use an Arduino? The KL25Z is well suited for this application, especially for its well form factor that it fits into Arduino shields! Which makes the wiring a little nicer.

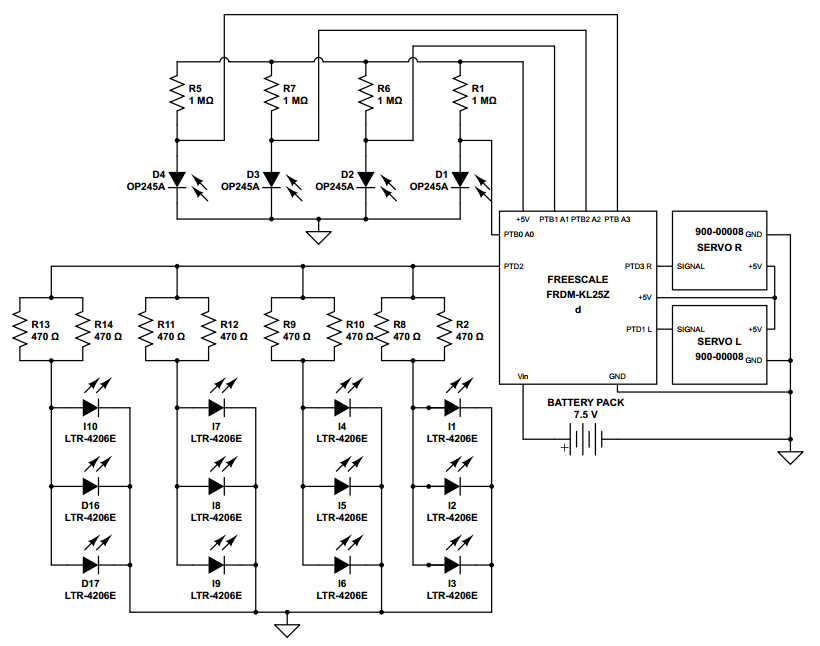


Showing most of the robot, all of the breadboards attached to the sides can be seen, and are connected to the analog ports on the shield, which just connect down the FRDM KL25Z, which is mounted below the shield, but above the metal frame. (USB cord stowed on robot for quick debugging purposes)

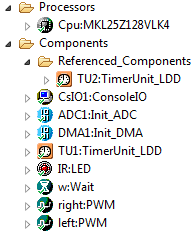


Each circuit board performs two functions, one combined function. IR Emitters and IR Photodiodes working together to create a proximity sensor.

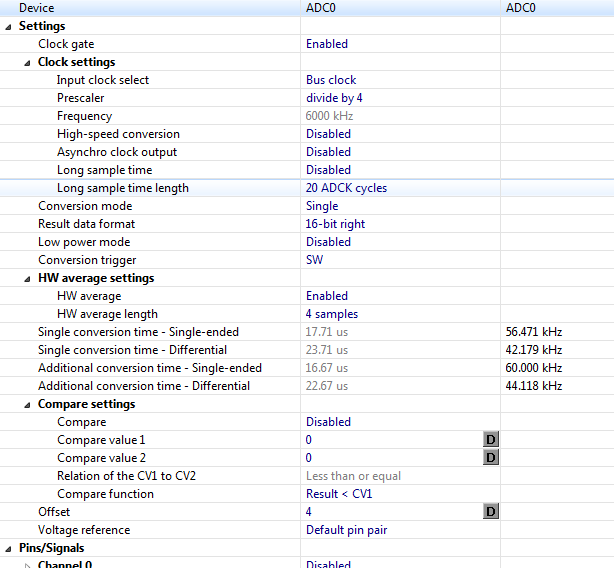
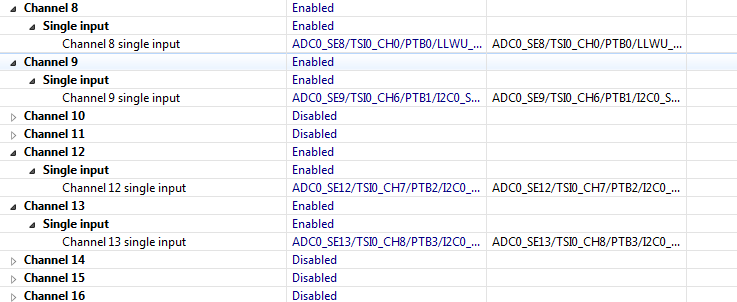
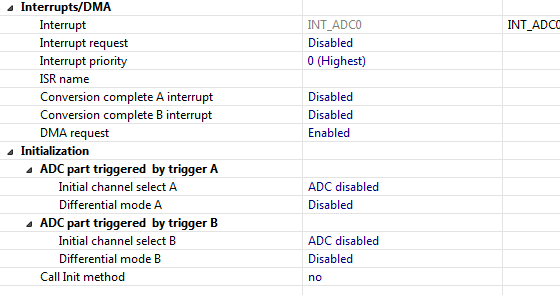




The brains of the robot is what resides in the code for the processor, which is written in C using CodeWarrior and Processor Expert. The first step to using CodeWarrior with Processor Expert is to pick the components and device drivers from the list of components. In the picture below is what is used on MouseBot. You’ll also see these be referenced quite a few times inside the actual program.



The next step is to change all of the properties of every component so that they will work with your processor and your actual components that are being controlled by the processor.

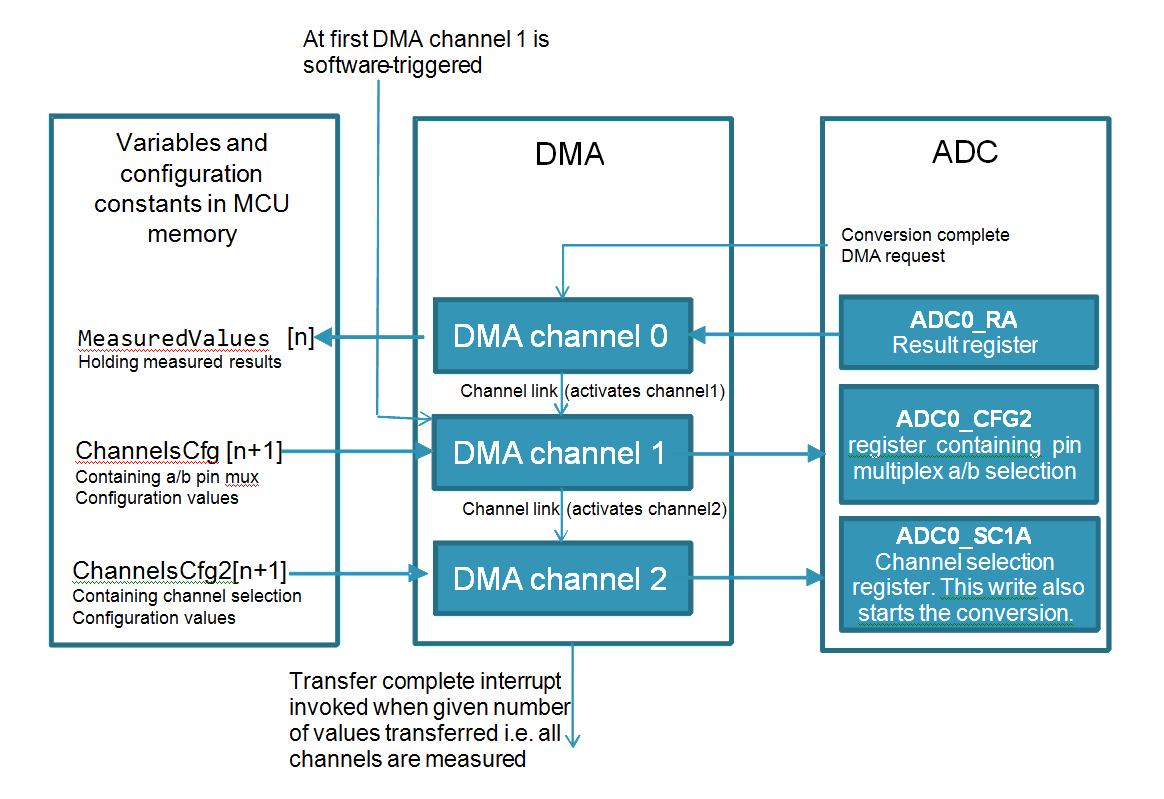


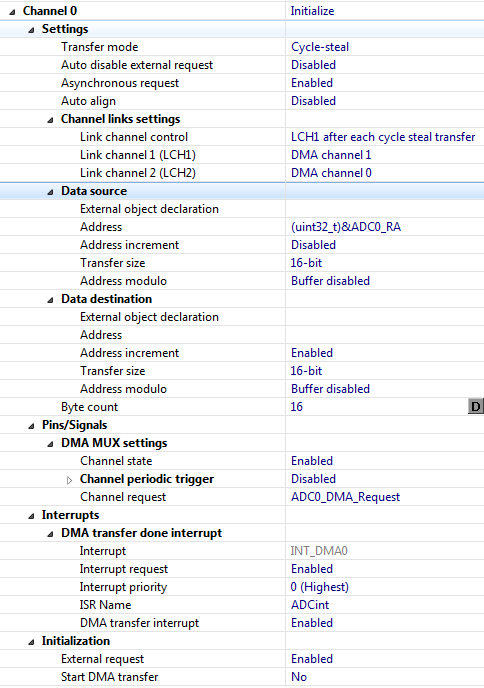
**Configuring the ADC to use the DMA**

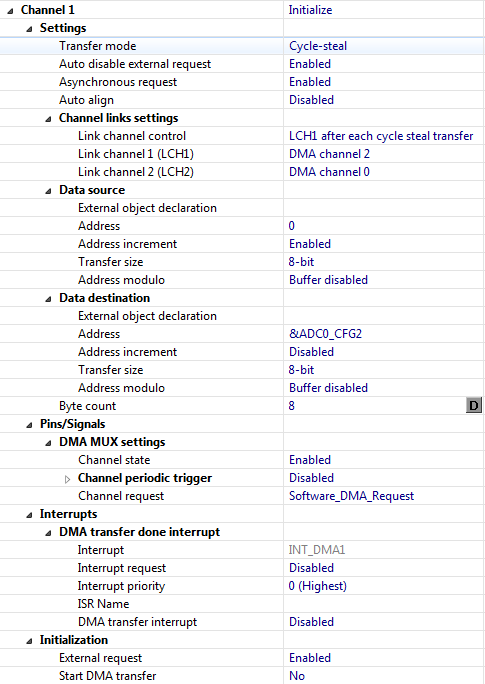
First we configure the ADC just as normal, selecting our resolution, averaging, timing, and channels. Then, on the bottom of the properties editor for the ADC, there is an important DMA request checkbox that must be enabled for the ADC to work the the DMA.

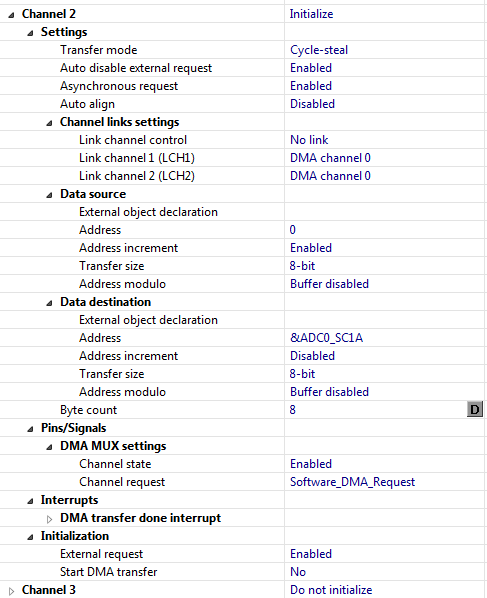
**DMA Properties**

DMA stands for Direct Memory Access, which is a simple concept, but difficult to understand at first, so hopefully this process flow diagram helps a bit.

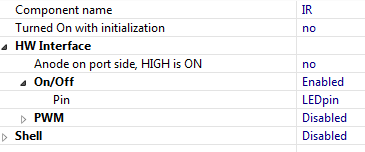
The DMA set up here uses 3 channels, one channel to get the value of the measured input, one channel to select the pin that the DMA wants (a number), and then another channel to select what channel the DMA wants the ADC to run on (A or B). In the next few pages it shown how the DMA is set-up for this task. The arrays that hold the pin and channel numbers that the DMA needs to check will be in the main source file (ProcessorExpert.c in these project files).





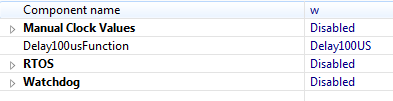


**IR LED**

The IR LED driver is a simple GPIO driver; it’s either on, or off. Here is the set up info for it anyways though.

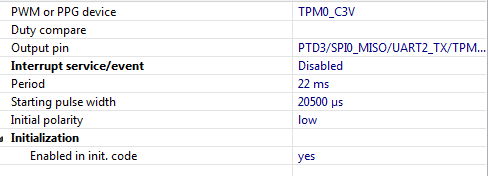
**Wait Component**

This is a small, but very necessary component in processor expert. This is also a very simple component, virtually just drog and droppable, but, because of the way that processor does it’s threading and timings, if you make your own wait function, it may not work as well. This is because this wait timer continues to do other threads while it works, making down time minimal for the processor, and making those data connections seamless. Here’s the set up for it!



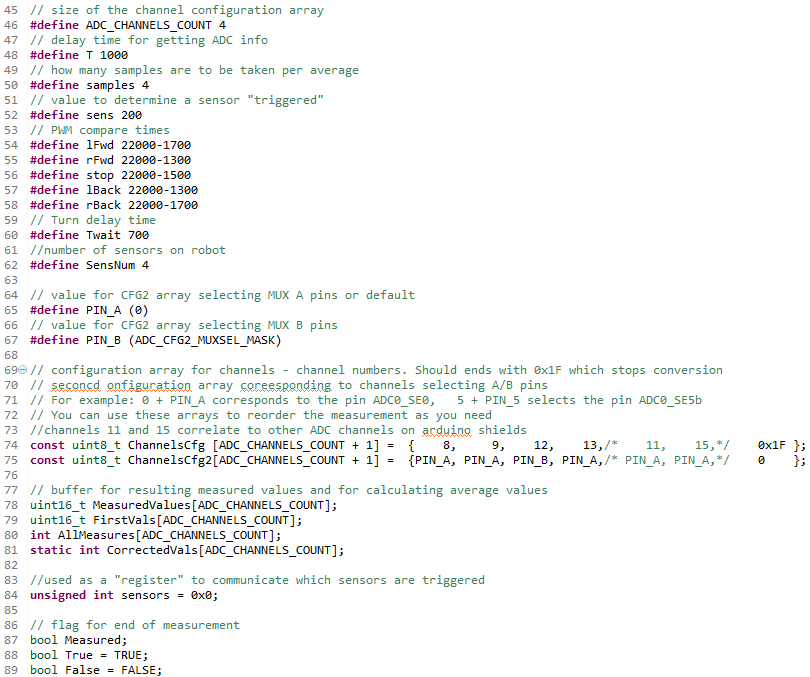
**PWM Components**

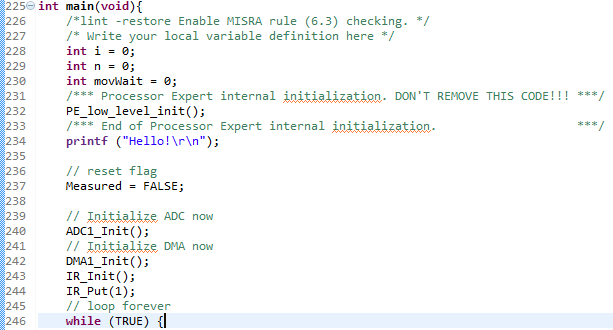
PWM components are for the two servos that move the entire car, now, these are the same set-ups for both, just the directions are different values for the motors in the code.

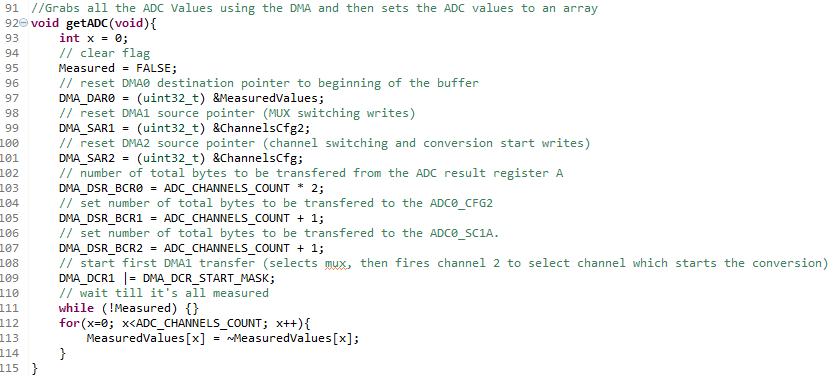


**The Actual Code**

First step to coding is defining your variables!! So, that’s shown here.

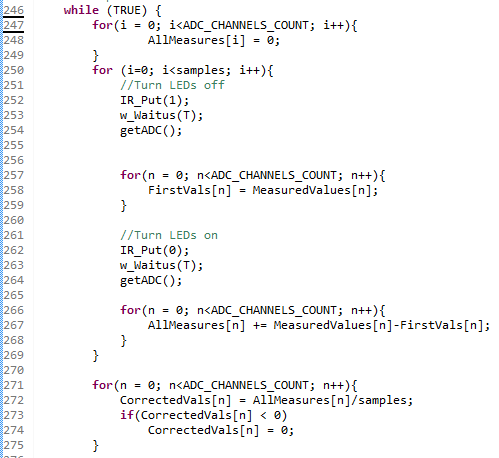


After the variables are defined, it’s time to put some logic in the program. Let’s start with initializing things!

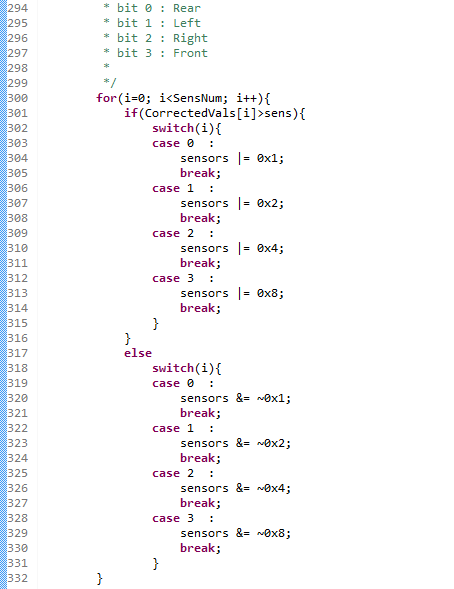
After initialization, we can start to get our ADC value samples and do some math with the values to calibrate and average our data with the outside world. First, we have to get our ADC Values, because this is going to be common thing that needs to be done, I took it out of the main function and made it its own function.

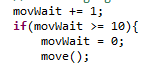
So now, all we have to do is call getADC() and our ADC values will be stored in the MeasuredValues array. After getting our data, we must then normalize it, and calibrate it. This can be done by taking more data! Shocking right?! First, let’s calibrate the data, this includes toggling the IR led on and off; first we take the value of the IR Phototransistors when the IR diodes are off, then we take the value of the IR Phototransistors when the IR diodes are on, then just subtract the two. The difference will tell you if there is an object in the way or not. The first step of this is to calibrate it so that you know how much IR light is being let in from sources other than MouseBot so that it can subtract that amount from the amount that it senses when the MouseBot’s IR lights are on.

In order to take the average, we get the calibrated result a few times, 4 in MouseBot’s programming, and then take the simple average formula to the 4 samples. One key concern with doing this is the types of variables used in the calculation of this, especially since there is floating point math involved.

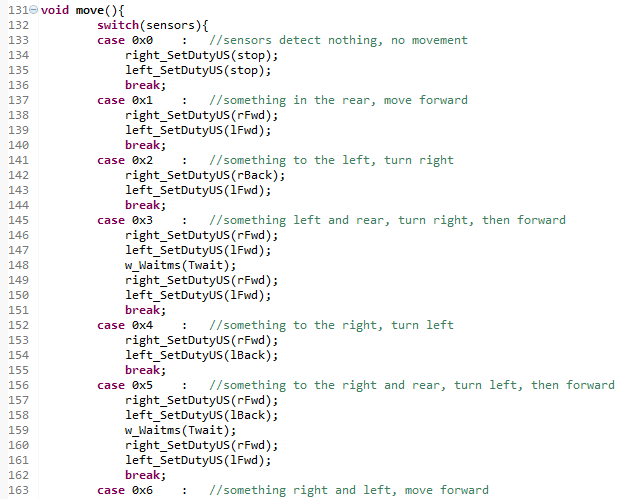


Now that we have normalized, averaged, and clean data, we should start comparing it to some values so that we know if a sensor has been tripped or not. There are many ways of doing this, but I thought it would be easiest to use a binary representation of a number, making the number act as a switch. It would be easier to visualize what’s happening here if C was able to interpret binary in the source files, but it can only do Hexadecimal, which still works!



Now that we know what sensors are detected, we need get the mouse out of that objects way! This is done by setting the duty cycle of the PWM components. To make MouseBot less jittery, I added a delay to when MouseBot gets told to move out of the way, every 10 cycles using a simple counter and if block.

The move() function that is called is what actually changes the duty cycle of the PWM components.



Results/Problems:

I could not have asked for better results, especially this being a prototype. I thought that the brute force method of a big case statement for every possible combination of detectors would be jittery trying to move the robot, but adding the delay functions solved that problem right away. Obviously though, if more than 4 sensors were used, making these case statements would be painful, and for that reason, another method of moving MouseBot is being developed.

Conclusion:

Using ARM and Processor Expert can most definitely get some getting used to, but after the learning curve, it cuts development time down significantly. I would like to see more of this program not only in class, but out of class as well.