Application Note:

The two main parts I focused the most on for project AWOL were antennas, and the arduino.

For the antenna what I realized is that sometimes the designs that you look up are unrealizable with the tools we have to work with. The problem I had with the design of the U antenna that I researched is that it required a specific probe feed which I could not realize. This is the reason our group went with the bow tie idea Brandon had. He had a fabricateable antenna that had a huge bandwidth. The reason I chose to look at the U-antenna is that it could have up to about 50% bandwidth. The 50% bandwidth is for S11 where the parameters are better than or equal to -10 dB. The challenging part about complex antenna design is figuring out to setup your ports for measurements and how to feed them. Typically you want 50 ohm lines at each port, but if you want high bandwidth those 50 ohm lines need to look like 50 ohm lines at a wide range of frequencies. Also, the best way to get the most accurate simulation results is to draw a boundary that covers your entire structure. Finally, the ports for simulation should be done using waveguide ports.
Another insight that I wanted to share about antenna design is that in order to mill/fabricate the antenna you need to make the file a gerber file. Using the software given to us from the antennas and radiation course(emPro) in UC Davis is not good enough to make the gerber layout which could be used for fabrication. However, one way to make a gerber file from a design in emPro would be to export the file and import it to ADS. The way to do this is to go to file, click export and export parts as an iges file. Then in ADS you go to schematic and hit layout. Then if you want to go to file an click import and you will be able to see a 2-D version of your layout. Then you will want to unclick the grid snap button in order to allign all the corners so the dimensions of the layout make sense. After the layout has been edited you can export the file as an gerber file which can be used in the milling machines to be fabricated. Another, way to be able to make gerber files is to use completly different software. You could use eagle, but the drawing tools are not that user friendly. Or another more useful tool would be to re-draw your schematic using HFSS.
For the arduino there are many good videos online that can show you how to implement codes, and specific commands for whatever you need to do. Also, in the window of a sketch there is a useful tab. If you click File, then go to examples you can find sample codes of many different uses of the microcontroller to get a template of how the code works. However, the arduino language works very similar to C, and is a relatively easy language to compile. In order to test the code you have written you need to upload the code onto your chip and test the results.
The arduino uno unfortunately did not have a built in Digital to Analog converter, which was a shame, because we could have saved some board space without the Resistor ladder DAC we ended up implementing. However, the plus of the arduino uno is that is has a built in ADC which we ended up taking advantage of for our project. The built in ADC of the arduino works by reading an analog input value(DC signal going into the arduino), and then partioning the value into a 10 bit sequence. If your project requires more bits then another microcontroller will be needed, but for the purpose of the project we used, we only needed 8-bits. The way we coded our 10-bit ADC is that we read the input going into the arduino and we set a constant bit number that was

less than 2^10. To determine the number of bits of voltage_in one can use the following equation.
N/(2^(number of bits)) = Vin/Vout
For our case the number of bits for the ADC was 10, and since the arduino uno can only output 0 or 5 volts(another limitation of the arduino uno) therefore Vout =5. Also, Vin will not be greater than V_out. The way the ADC works is that we set a threshold value(in bits) based off the equation above for N, and if we see a value greater than it we say digitalwrite(). DigitalWrite() will make your variable either high or low, and if you wanted to make a continuous square wave you just need to make a loop of setting a pin from high to low with enough delay in between to see the jump.
The arduino uno was used in our project to control our VCOs, the ADC on our receiver, and to verify our system recorded the correct sequence. By being able to control the bit code the arduino outputs we could control the V_tune going into our VCO so we can control our mixers on both the transmitter and the receiver. Once the arduino recognizes that we had the right code, we programmed the arduino to output a 0-5V square wave to sound off our buzzer. The 0-5 square wave is essentially turning a pin off and on with delay. However, the delay determines how loud the buzzer sounds off, and was determined through trial and error.
Another limitation of the arduino uno was the difficulty of loading a brand new chip. In order to "install" or burn a bootloader onto an arduino atmega chip you need a chip that already works. So if you break a chip, and order a new atmega chip you will not be able to simply just go and load your code again, but need to burn a bootloader onto your new atmega chip.
Another important tip to remember is to go over the datasheet of the actual chip of the atmega chip before wiring and applying a voltage to your system. The pinout of the chip does not match up with the pins of the arduino uno board. This is important to look at so you do not damage the chip. If the arduino uno does not have all the requirements you need for your specifications the you will need to look into using a more powerful microprocessor such as the Due.