LAB Assignment #4 for ECE 525

Description: Process PND_c into PND_{co} and $modPND_{co}$

1) This lab adds to the code you created for in the previous lab and implements the *Offset* and *Modulus* operations that are part of the HELP algorithm.

2) From the previous lab, you have an array of floating point values, PND_c, of size 2048. Convert the PND_c to PND_{co} as described by the pseudo-code and then store the values into a 5th 2-D floating point array of 2048 elements (beyond the PNR, PNF, PND and PND_c arrays) called PND_{co}. The following pseudo-code describes a method that will add a random offset to each the elements in the new array PND_{co}. Later, when we implement the server portion of the authentication operation, you will replace the random offsets with values computed and transmitted by the server to the token. Use the *LFSR_11_A_bits_low* routine provided in the previous lab in the LFSR calls below. Note: random offset is restricted to a value between 0 and *Modulus*/2

```
void ComputePNDco(int max_PNDiffs, float PNDc[max_PNDiffs], float PNDco[max_PNDiffs],
   int LFSR_seed, int Modulus)
   {
   Divide the parameter Modulus by 32 and store this floatig point constant in a variable
      called offset_delta
   for each i in PNDc,
      if i is 0
         LFSR_11_A_bits_low: Initialize LFSR with LFSR_seed and get first LFSR_val
      else
         LFSR_11_A_bits_low: Get next LFSR_val
      Zero out bits 4 to 31 in LFSR_val using a mask, i.e., keep only the low order 4 bits
      Multiply LFSR_val by offset_delta (round to 4 bits of binary precision) and add to PNDc
      Store result in PNDco
   }
```

3) The modulus operator is then applied to the PND_{co} to create $modPND_{co}$. The PND_{co} are floating point values so you'll need to compute the Modulus of the PND_{co} using the following pseudo-code

```
void ComputeModulus(int max_PNDiffs, float PNDco[max_PNDiffs], float modPNDco[max_PNDiffs], int Modulus)
   {
   for each PNDco
      Copy PNDco into modPNDco, rounding to 4 binary digits
      while(1)
         {
         if modPNDco is less than 0
            Add Modulus to modPNDco
         else if modPNDco is greater than 0
            Subtract Modulus from modPNDco
         else
            break
         }
      Round final modPNDco to 4 binary digits
   }
```

In your trials, use a LFSR_seed of 0 and a *Modulus* of 20 for the parameters to these routines. Use the verifier_regeneration program from the previous lab.

4) Create a lab report that includes a description of this lab. Include a copy of your *ComputePNDco* and *ComputeModulus* C code. Run your code on the Zybo board and generate a histogram graph using your favorite software (matlab) of the modPNDco array.