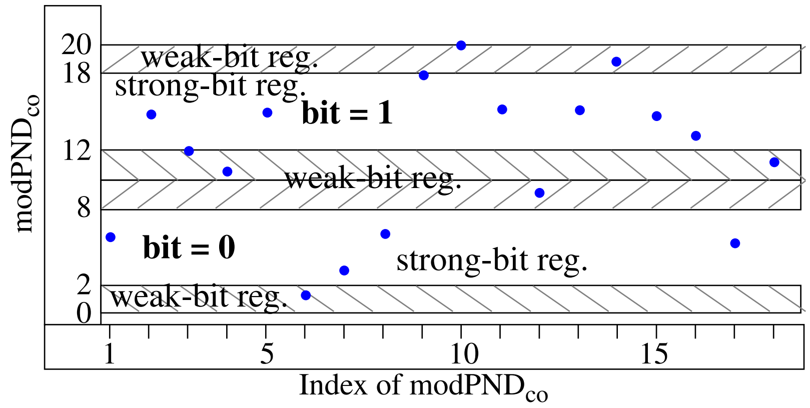
LAB Assignment #5 for ECE 525

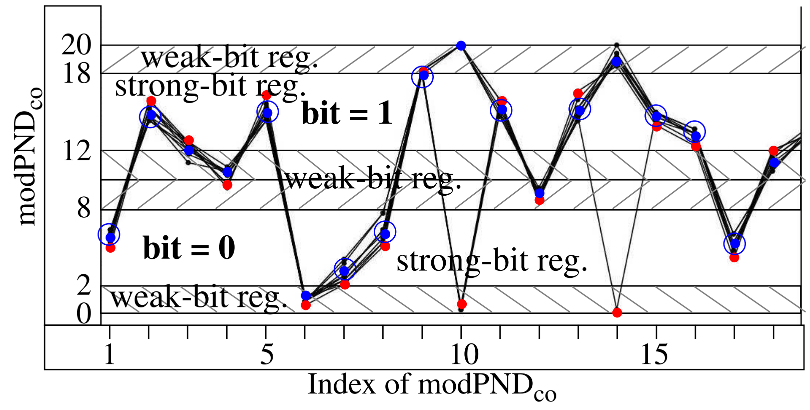
Description: Process modPND$_{co}$ into strong bitstrings

1) This lab adds to the code you created for in the previous lab and implements HELP's Single Helper Data scheme as described in the screencasts. After the last lab, you have an array of floating point values, modPNDco, of size 2048. Write a routine that computes the strong bitstring and helper data bitstring described as follows and in the HELP screencasts.

The modPND$_{co}$ values can be represented as a graph shown below. The x-axis plots the index of consecutative values in your array while the y-axis represents the computed values. In order to avoid bit flip errors, you need to apply the HELP Margining technique that excludes specific modPNDco from participating in the bitstring generation process. The margin in the following illustration is set to 2. The margin creates *weak-bit regions* around the bit-flip lines 0, 10 and 20. modPND$_{co}$ that fall on or within these *weak-bit regions* have a higher probability of introducing a bit flip error during regeneration than those in the *strong-bit regions*.



For example, if we were to re-collect the modPND$_{co}$ under other environmental conditions, i.e., with different supply voltage values and/or temperature conditions, the values will *shift* as shown by the following set of curves. The red dots represent a 'worst-case' shift that occurs to each modPND$_{co}$. The vertical shift in modPND$_{co}$ that are close to the bit-flip lines can result in the modPND$_{co}$ value being re-classified from a '1' to a '0' or vise versa (identify where this occurs in the graph).



Therefore, modPND$_{co}$ that fall within the *weak-bit regions* are to be skipped when generating the bitstring. For example, the *strong* bitstring (SBS) generated by processing the modPND$_{co}$ shown by this graph is as follows (NOTE: values that fall ON the line are considered weak):

0	**1**	**1**	**0**	**0**	**1**	**1**	**1**	**1**	**1**	**0**

So only 11 bits of the possible 18 are actually used. A **helper data bitstring** is also generated that identifies which bits are classified as *strong-bits* (blue points) during the enrollment process.

1	1	0	0	1	0	1	1	1	0	1	0	1	0	1	1	1	0
1				5					10					15			

The helper data bitstring is used during regeneration to ‘select’ the modPND$_{co}$ values to be used for generating the bitstring, i.e., **the margins are NOT used during regeneration**.

The following pseudo-code can be modified to use of two functions for inserting bits into and extracting bit from the array of unsigned char bytes used to store the strong bitstring and helper data bitstring.

```
int GetBitFromByte(char byte, int bit_pos)
   { return ((byte & (1 << bit_pos)) == 0) ? 0 : 1; }

void SetBitInByte(char *byte_ptr, int bit_val, int bit_pos)
   { *byte_ptr = (bit_val == 0) ? (*byte_ptr) & ~(1 << bit_pos) : (*byte_ptr) | (1 << bit_pos); }


int SingleHelpBitGen(int max_PNDiffs, float fmodPNDco[max_PNDiffs], unsigned char SBS[max_PNDiffs/8],
   unsigned char SHD[max_PNDiffs/8], unsigned short Margin, unsigned short Modulus)
   {
   set floating point threshold to Modulus/2
   set SBS_num_bits to 0
   for each fmodPNDco
      {
      Decide if current fmodPNDco generates a 0 or 1 bit
      Classify the current fmodPNDco as a strong or weak bit
      Store the helper data bit into the SHD array
      For those fmodPNDco classifed as strong, add the strong bit to the SBS array
      }

   return the number of strong bits
   }
```

4) Create a lab report that includes a description of this lab. Include a copy of your *ComputePNDco* and *ComputeModulus* C code. Use a Modulus of 20 and a Margin of 3 in your trial runs. Run your code on the Zybo board and generate the strong bitstring and helper data bitstring and include both in your report.