HISTO 'C' Program

Sftwre

```
1  //
   ================================================================================
   ====================
2  //
   ================================================================================
   ====================
3  // ********************************************** Histo.c
   **********************************************
4  // Does Histo.c load the data file into PN-BRAM?? Yes, line 154
   ================================================================================
   ====================
5  // Does Histo.c need to pass data to hardware??
   ================================================================================
   ====================
6
7  #include "common.h"
8
9
10 // What are the main imports & exports
   ================================================================================
   ========================
11 //
   ================================================================================
   ========================
12 //
   ================================================================================
   ========================
13 // C algorithm of the function carried out in the hardware
14
15 void ComputeHisto(int max_vals, int num_vals, short *vals, short LV_bound, short
   HV_bound,
16                   short DIST_range, short precision_scaler, short *software_histo)
17 {
18    short LV_addr, HV_addr, LV_set, HV_set;
19    int PN_num, bin_num, HISTO_ERR;
20    short smallest_val;
21    short dist_cnt_sum;
22    int dist_mean_sum;
23    short temp_val;
24    short range;
25
26 // Initialize variables.
27    HISTO_ERR = 0;
28    dist_mean_sum = 0;
29
30 // Clear out the counts in the distribution bins.
31    for ( bin_num = 0; bin_num < DIST_range; bin_num++ )
32       software_histo[bin_num] = 0;
33
34 // Find smallest value. Then obtain the integer portion (low order 4 bits of the
   shorts are assumed to be part of the
35 // fractional component by the hardware -- fixed point floats).
36    for ( PN_num = 0; PN_num < num_vals; PN_num++ )
37       if ( PN_num == 0 )
38          smallest_val = vals[PN_num];
39       else if ( smallest_val > vals[PN_num] )
40          smallest_val = vals[PN_num];
41    smallest_val /= precision_scaler;
42
43 // Construct the histogram and compute the mean
44    for ( PN_num = 0; PN_num < num_vals; PN_num++ )
45       {
46
47 // Add current val to sum for mean calc.
```

1st into Array
2nd into BRAM (Hardware)

refs to... GPIO_BASE_ADDR??

histo_arr_out = hardware HIST

software_histo = 'c' HISTO prgm.

(externally) of the 'c' prgm?:

in ← screen → out
in ← data file
in ← BRAM → out

Boots Connects to Top??

```
48          dist_mean_sum += (int)vals[PN_num];
49
50    // Adjust integer portion of vals by subtracting smallest value in the
      distribution.
51          temp_val = vals[PN_num]/precision_scaler - smallest_val;
52
53    //printf("%d) temp_val %d\n", PN_num, temp_val);
54
55    // Sanity check.
56          if ( temp_val >= DIST_range )
57             HISTO_ERR = 1;
58
59          software_histo[temp_val]++;
60          }
61
62    // Sweep the histogram and record the address where the lower and higher bounds
      are exceeded.
63       LV_addr = 0;
64       HV_addr = 0;
65       LV_set = 0;
66       HV_set = 0;
67       dist_cnt_sum = 0;
68       for ( bin_num = 0; bin_num < DIST_range; bin_num++ )
69          {
70          dist_cnt_sum += software_histo[bin_num];
71
72    // As soon as the is satisfied the first time, stop updating it.
73          if ( LV_set == 0 && dist_cnt_sum >= LV_bound )
74             {
75             LV_addr = bin_num;
76             LV_set = 1;
77             }
78
79    // Keep updating until the bound is exceeded than stop.
80          if ( dist_cnt_sum <= HV_bound )
81             {
82             HV_addr = bin_num;
83             HV_set = 1;
84             }
85          }
86       range = HV_addr - LV_addr + 1;
87
88    // Error check
89       if ( LV_set == 0 || HV_set == 0 )
90          HISTO_ERR = 1;
91
92       if ( HISTO_ERR == 1 )
93          printf("ERROR: ComputeHisto(): Histo error!\n");
94
95       printf("Software Computed Stats: Smallest Val %d\tLV_addr %d\tHV_addr %d\tMean
         %.4f\tRange %d\n",
96          smallest_val, LV_addr, HV_addr,
            (float)(dist_mean_sum/num_vals)/precision_scaler, (int)range);
97       fflush(stdout);
98
99       return;
100      }
101
102
103   //
      ================================================================================
      ====================
104   //
      ================================================================================
```

```
//=====================
// Read integer data from a file and store it in an array.

int ReadData(int max_string_len, int max_data_vals, char *infile_name, short
*data_arr_in)
   {
   char line[max_string_len], *char_ptr;
   float temp_float;
   FILE *INFILE;
   int val_num;

   if ( (INFILE = fopen(infile_name, "r")) == NULL )
      { printf("ERROR: ReadData(): Could not open %s\n", infile_name);
      fflush(stdout);  exit(EXIT_FAILURE); }

   val_num = 0;
   while ( fgets(line, max_string_len, INFILE) != NULL )
      {

   // Find the newline and eliminate it.
      if ((char_ptr = strrchr(line, '\n')) != NULL)
         *char_ptr = '\0';

   // Skip blank lines
      if ( strlen(line) == 0 )
         continue;

   // Sanity check
      if ( val_num >= max_data_vals )
         { printf("ERROR: ReadData(): Exceeded maximum number of vals %d!\n",
         max_data_vals); fflush(stdout); exit(EXIT_FAILURE); }

   // Read and convert value into an integer
      if ( sscanf(line, "%f", &temp_float) != 1 )
         { printf("ERROR: ReadData(): Failed to read an float value from file
         '%s'!\n", line); fflush(stdout); exit(EXIT_FAILURE); }

   // Sanity check
      if ( (int)(temp_float*16) > MAX_SHORT_POS || (int)(temp_float*16) <
      MAX_SHORT_NEG )
         { printf("ERROR: ReadData(): Scaled float (by 16) larger than max or
         smaller than min value for short %d!\n", data_arr_in[val_num]);
         fflush(stdout); exit(EXIT_FAILURE); }

      data_arr_in[val_num] = (int)(temp_float*16);

      val_num++;
      }

   fclose(INFILE);

   return val_num;
   }

//=====================================================================
// Load the data from the data arry into the secure BRAM

void LoadUnloadBRAM(int max_string_len, int max_vals, int num_vals, int
```

only 1st 12 bits

1st read data file into array for 'C' use

not same as comm.h.

Keep no change

eos end of string null character tells while loop to stop

if true continue

if true, continue

if true, stop / false, continue

if true stop / false continue

if true stop / false increment & continue

Insert here??
Section on 'Difference'
Different the data before or after, loading into BRAM?? would require 2 load ops.

val_num_UPR = 6144 - 8192

val_num_Lwr = 4096 - 6143

Different these 2, store results in new array

Then, move/copy same data to BRAM for the hardware to be able to use (VHDL)

```
load_unload, short *IOData,
   volatile unsigned int *CtrlRegA, volatile unsigned int *DataRegA, int ctrl_mask)
   {
   int val_num, locked_up;

   for ( val_num = 0; val_num < num_vals; val_num++ )
      {

// Sanity check
      if ( val_num >= max_vals )
         { printf("ERROR: LoadUnloadBRAM(): val_num %d greater than max_vals
         %d\n", val_num, max_vals); exit(EXIT_FAILURE); }

// Four step protocol
// 1) Wait for 'stopped' from hardware to be asserted
//printf("LoadUnloadBRAM(): Waiting 'stopped'\n"); fflush(stdout);
      locked_up = 0;
      while ( ((*DataRegA) & (1 << IN_SM_HANDSHAKE)) == 0 )
         {
         locked_up++;
         if ( locked_up > 10000000 )
            {
            printf("ERROR: LoadUnloadBRAM(): 'stopped' has not been asserted for
            the threshold number of cycles -- Locked UP?\n");
            fflush(stdout);
            locked_up = 0;
            }
         }

// 2) Put data into GPIO (load) or get data from GPIO (unload). Assert 'continue'
for hardware
// Put the data bytes into the register and assert 'continue' (OUT_CP_HANDSHAKE)
//printf("LoadUnloadBRAM(): Reading/writing data and asserting 'continue'\n");
fflush(stdout);
      if ( load_unload == 0 )
         *CtrlRegA = ctrl_mask | (1 << OUT_CP_HANDSHAKE) | (0x0000FFFF &
         IOData[val_num]);

// When 'stopped' is asserted, the data is ready on the output register from the
PNL BRAM -- get it.
      else
         {
         IOData[val_num] = (0x0000FFFF & *DataRegA);
         *CtrlRegA = ctrl_mask | (1 << OUT_CP_HANDSHAKE);
         }

//printf("%d\tData value written or read %d\n", val_num, IOData[val_num]);
fflush(stdout);

// 3) Wait for hardware to de-assert 'stopped'
//printf("LoadUnloadBRAM(): Waiting de-assert of 'stopped'\n"); fflush(stdout);
      while ( ((*DataRegA) & (1 << IN_SM_HANDSHAKE)) != 0 );

// 4) De-assert 'continue'. ALSO, assert 'done' (OUT_CP_LM_ULM_DONE)
SIMULTANEOUSLY if last word to inform hardware.
//printf("LoadUnloadBRAM(): De-asserting 'continue' and possibly setting
'done'\n"); fflush(stdout);
      if ( val_num == num_vals - 1 )
         *CtrlRegA = ctrl_mask | (1 << OUT_CP_LM_ULM_DONE);
      else
         *CtrlRegA = ctrl_mask;
      }

// Handle case where 'num_vals' is 0.
```

address of 'c' array } whole 4,096 element set

address of GPIO Data window

address of ctrl GPIO Blo

16 bit wide

OK

Should not need to change for Lab #1 (I think)

How to communicate with Top.vhd.

Send PN data from data file

No change

16 bits data ?

seen before mask ??

control flags handshake info not just data txfr

Retrieves smaller Histo from Cmd

No change

```
211       if ( num_vals == 0 )
212          *CtrlRegA = ctrl_mask | (1 << OUT_CP_LM_ULM_DONE);
213
214    // De-assert 'OUT_CP_LM_ULM_DONE'
215       *CtrlRegA = ctrl_mask;
216
217       fflush(stdout);
218
219       return;
220       }
221
222
223    //
       ================================================================================
       ====================
224    //
       ================================================================================
       ====================
225    //
       ================================================================================
       ====================
226
227    int main(int argc, char *argv[])
228       {
229       volatile unsigned int *CtrlRegA;
230       volatile unsigned int *DataRegA;
231       unsigned int ctrl_mask;
232
233       char infile_name[MAX_STRING_LEN];
234       char outfile_name[MAX_STRING_LEN];
235
236       short *data_arr_in;
237       short *histo_arr_out;
238       short *software_histo;
239       int num_vals;
240       int load_unload;
241
242       int precision_scaler = 16;
243
244       struct timeval t0, t1;
245       long elapsed;
246
247    //
       ================================================================================
       ==================================
248    // COMMAND LINE
249       if ( argc != 2 )
250          {
251          printf("ERROR: LoadUnload.elf(): Datafile name (test_data_10vals.txt)\n");
252          return(1);
253          }
254
255       sscanf(argv[1], "%s", infile_name);
256
257    // Open up the memory mapped device so we can access the GPIO registers.
258       int fd = open("/dev/mem", O_RDWR|O_SYNC);
259
260       if (fd < 0)
261          { printf("ERROR: /dev/mem could NOT be opened!\n"); exit(EXIT_FAILURE); }
262
263    // Add 2 for the DataReg (for an offset of 8 bytes for 32-bit integer variables)
264       DataRegA = mmap(0, getpagesize(), PROT_READ|PROT_WRITE, MAP_SHARED, fd,
       GPIO_0_BASE_ADDR);
265       CtrlRegA = DataRegA + 2;
```

Should be no change in LM_ULM so long as data is packed in CtrlRegA & DataRegA.

for passing data file as argument.

status register / raw data } GPIO comms w/ TOP.VHD

DataRegA = mmap(...)
CtrlRegA = DataRegA + 2 // + 2 Bytes = I want 16 bits is length of our PN's

Probably exists in Linux OS on Zynq Boot, could do a 'find' to prove...

defined in common.h.

Memory map

```
266
267   // Allocate arrays
268      if ( (data_arr_in = (short *)calloc(sizeof(short), MAX_DATA_VALS)) == NULL )
269         { printf("ERROR: Failed to calloc data 'data_arr_in' array!\n");
           exit(EXIT_FAILURE); }
270      if ( (histo_arr_out = (short *)calloc(sizeof(short), MAX_HISTO_VALS)) == NULL )
271         { printf("ERROR: Failed to calloc data 'histo_arr_out' array!\n");
           exit(EXIT_FAILURE); }
272      if ( (software_histo = (short *)calloc(sizeof(short), MAX_HISTO_VALS)) == NULL )
273         { printf("ERROR: Failed to calloc data 'histo_arr_out' array!\n");
           exit(EXIT_FAILURE); }
274
275   // Read the data from the input file
276      num_vals = ReadData(MAX_STRING_LEN, MAX_DATA_VALS, infile_name, data_arr_in);
277
278   // Set the control mask to indicate enrollment.
279      ctrl_mask = 0;
280
281   //
      ============================================================================
282   // Software computed values. Hardware reports mean WITH 4 bits of precision but
      range using ONLY the integer portion.
283      gettimeofday(&t0, 0);
284      ComputeHisto(MAX_DATA_VALS, num_vals, data_arr_in, (short)LV_BOUND,
         (short)HV_BOUND, (short)DIST_RANGE, precision_scaler,
285        software_histo);
286      gettimeofday(&t1, 0); elapsed = (t1.tv_sec-t0.tv_sec)*1000000 +
         t1.tv_usec-t0.tv_usec;
287      printf("\tSoftware Runtime %ld us\n\n", (long)elapsed);
288   //
      ============================================================================
289
290   // Do a soft RESET
291      *CtrlRegA = ctrl_mask | (1 << OUT_CP_RESET);
292      *CtrlRegA = ctrl_mask;
293      usleep(1000);
294
295   // Wait for the hardware to be ready -- should be on first check.
296      while ( ((*DataRegA) & (1 << IN_SM_READY)) == 0 );
297
298   // Start clock
299      gettimeofday(&t0, 0);
300
301   // Start the VHDL Controller
302      *CtrlRegA = ctrl_mask | (1 << OUT_CP_START);
303      *CtrlRegA = ctrl_mask;
304
305   // Controller expects data to be transferred to the BRAM as the first operation.
306      load_unload = 0;
307      LoadUnloadBRAM(MAX_STRING_LEN, MAX_DATA_VALS, num_vals, load_unload,
         data_arr_in, CtrlRegA, DataRegA, ctrl_mask);
308
309   // Data transfer in time
310      gettimeofday(&t1, 0); elapsed = (t1.tv_sec-t0.tv_sec)*1000000 +
         t1.tv_usec-t0.tv_usec;
311      printf("\tHardware Transfer In time %ld us\n\n", (long)elapsed);
312
313   // Start clock
314      gettimeofday(&t0, 0);
315
316   // Wait for 'stopped' to be asserted by hardware. When this occurs, histogram FSM
      is finished and its ready to transfer
317   // data out.
318      while ( ((*DataRegA) & (1 << IN_SM_HANDSHAKE)) == 0 );
```

**** ADD ARRAY for PM_diffs **** new: data_diffs_arr_in

#vals

src values — must point to diff_data_arr_in

out out.

Ok to keep if control scan

'c' initiated reset external output to PL side

into position for output to place '1'

(out) to *CtrlRegA is where TOP.VHD takes as (Input).

1 2 3 4 5 6 7 8 — # arguments, different in common.h

state machine.

```
// Approx. runtime of hardware excluding I/O
   gettimeofday(&t1, 0); elapsed = (t1.tv_sec-t0.tv_sec)*1000000 +
   t1.tv_usec-t0.tv_usec;
   printf("\tHardware Runtime %ld us\n\n", (long)elapsed);

// Check for a HISTO error
   if ( ((*DataRegA) & (1 << IN_SM_HISTO_ERR)) == 1 )
      { printf("ERROR: Histogram error!\n"); exit(EXIT_FAILURE); }

// Start clock
   gettimeofday(&t0, 0);

// After computing the histogram, Controller expects to transfer histogram memory
and distribution parameters back to C program.
   load_unload = 1;
   LoadUnloadBRAM(MAX_STRING_LEN, MAX_HISTO_VALS, MAX_HISTO_VALS, load_unload,
   histo_arr_out, CtrlRegA, DataRegA, ctrl_mask);

// Data transfer out time
   gettimeofday(&t1, 0); elapsed = (t1.tv_sec-t0.tv_sec)*1000000 +
   t1.tv_usec-t0.tv_usec;
   printf("\tHardware Transfer Out time %ld us\n\n", (long)elapsed);

//                                    — no change —
// Print out the histogram. The mean and range are the last two values (of the
2048).
   int i;
   printf("HISTOGRAM VALUES:\n");
   for ( i = 0; i < MAX_HISTO_VALS - 2; i++ )
      {
      printf("(%4d) %3d  ", i, histo_arr_out[i]);
      if ( ((i+1) % 10) == 0 )
         printf("\n");

// Sanity check. Both the hardware and software histogram should be identical.
      if ( histo_arr_out[i] != software_histo[i] )
         {
         printf("ERROR: Mismatch between hardware %d and software %d histos at
         index %d\n",
            histo_arr_out[i], software_histo[i], i); exit(EXIT_FAILURE);
         }
      }
   printf("\n\n");


// Write out an xy file with histogram data for histogram plotting with R.
Assumes the input file is "yyy.txt"
   FILE *OUTFILE;
   char temp_str[MAX_STRING_LEN];

   strcpy(temp_str, infile_name);
   temp_str[strlen(infile_name) - 3]  = '\0';

   strcpy(outfile_name, "Output_");
   strcat(outfile_name, temp_str);
   strcat(outfile_name, "xy");

   if ( (OUTFILE = fopen(outfile_name, "w")) == NULL )
      { printf("ERROR: ReadData(): Could not open %s\n", outfile_name);
      exit(EXIT_FAILURE); }
   for ( i = 0; i < MAX_HISTO_VALS - 2; i++ )
      fprintf(OUTFILE, "%d\t%d\n", i, histo_arr_out[i]);
```

external out = screen

change in common.h

histo_arr_out[i]

Hardware Histogram.

external out, file struct.

must change in common.h

```
374       fclose(OUTFILE);
375
376
377       printf("Hardware Computed Mean %.4f\tRange %d\n",
378          (float)histo_arr_out[MAX_HISTO_VALS-2]/precision_scaler,
          histo_arr_out[MAX_HISTO_VALS-1]);
379    //
       ================================================================================
380
381    // Check if Controller returned to idle
382       if ( ((*DataRegA) & (1 << IN_SM_READY)) == 0 )
383          { printf("ERROR: Controller did NOT return to idle!\n");
          exit(EXIT_FAILURE); }
384
385       return 0;
386       }
```