一. Code 解說:

```
void histogram_map(int in[INPUT_SIZE], int hist[VALUE_SIZE]) {
#pragma HLS DEPENDENCE variable=hist intra RAW false
    for(int i = 0; i < VALUE_SIZE; i++) {
#pragma HLS PIPELINE II=1
        hist[i] = 0;
    }
  int old = in[0];
  int acc = 0;
  for(int i = 0; i < INPUT_SIZE; i++) {
#pragma HLS PIPELINE II=1
    int val = in[i];
    if(old == val) {
      acc = acc + 1;
    } else {
      hist[old] = acc;
      acc = hist[val] + 1;
    }
    old = val;
  }
  hist[old] = acc;
}
```

首先這個 histogram_map 函式，因為我想要更優化直方圖計算，所以我將其分為兩階段來達到更多並行性，而這個 histogram_map 函數的輸入矩陣就是 hist 陣列中的一個分區數據，將輸入數據分成兩個分區，分別存儲在 inputA 和 inputB 陣列中。 它使用 histogram_map 函數計算每個分區的直方圖，然後將其存儲在 hist1 和 hist2 陣列中。 這兩個陣列被輸入到 histogram_reduce 函數中。該函數將它們合併後並將結果存儲在 hist 陣列中，其中合併的 hist 陣列是頂層直方圖函數的最終輸出。首先這個 DEPENDENCE PRAGMA 是因為每次反覆運算迴圈中，

我們讀取另一個 hist 陣列中另一個位置 x1 中陣列。 x0 和 x1 都取決於輸入值，並可以取任何值。 因此我們考慮到綜合成電路時最壞的情況，如果 x0 = x1，則在前一個寫入完成前，位置 x1 處的讀取無法開始。 因此，我們必須在讀寫之間進行切換。再來使用 PIPELINE PRAGMA 在展開內部迴圈時能夠得到更少的時間，再來下面的迴圈可以看到流程中的'loop'包含存儲 out[]陣列的輸出存儲部分，流程中的迴圈部分僅包含一個存儲累加值的寄存器和只寫的輸出記憶體。

```
void histogram_reduction(int hist1[VALUE_SIZE], int hist2[VALUE_SIZE], int output[VALUE_SIZE]) {
  for(int i = 0; i < VALUE_SIZE; i++) {
#pragma HLS PIPELINE II=1
    output[i] = hist1[i] + hist2[i];
  }
}
```

Histogram_reduce 函數實現了模式中的「還原」部分。 它將分區數據的直方圖作為輸入，並通過將每個直方圖的計數相加，將它們組合成完整的直方圖。因為我們只有兩個處理物件，因此將兩個輸入陣列 hist1 和 hist2 合併。 這可以很容易的擴展以處理更多的元素。

```
void histogram(int inputA[INPUT_SIZE], int inputB[INPUT_SIZE], int hist[VALUE_SIZE]){
  #pragma HLS DATAFLOW
  int hist1[VALUE_SIZE];
  int hist2[VALUE_SIZE];

  histogram_map(inputA, hist1);
  histogram_map(inputB, hist2);
  histogram_reduction(hist1, hist2, hist);
}
```

在 histogram 函數中使用 dataflow 指令來達到流水線設計。這邊有三個模組：兩個 histogram_map 函數和一個 histogram_reduce 函數。因為兩個 histogram_map 處理的數據相互獨立，所以可以同時執行。 Histogram_reduce 函數處理過程必須在 histogram_map 處理完成後才開始。 因此，dataflow 指令本質上是創建了一個兩階段的管道。 第一階段執行 histogram_map 函數，而第二階段執行 histogram_reduce 函數。

二. 執行結果:

Modules && Loops	Issue Type	Violation Type	Distance	Slack	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM	DSP	FF	LUT	URAM
histogram				-	1066	1.066E4	-	1067	-	dataflow	8	0	395	903	0
histogram_map3				-	1066	1.066E4	-	1066	-	no	0	0	170	383	0
histogram_map				-	1066	1.066E4	-	1066	-	no	0	0	170	383	0
histogram_reduce				-	35	350.000	-	35	-	no	0	0	53	105	0