How to use HLS optimization directives for Image Histogram Equalization

Daniele Bagni,
DSP Specialist for EMEA

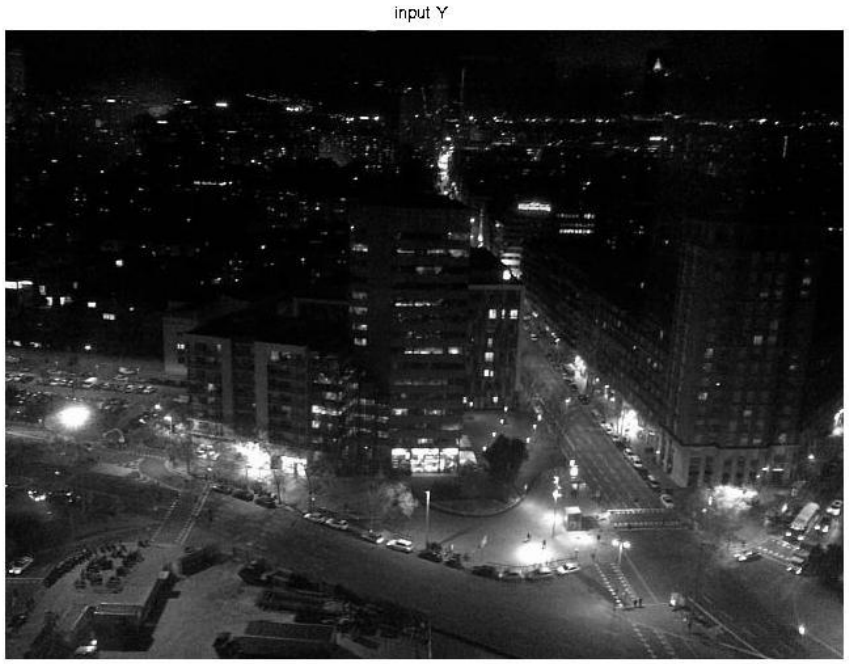
Outline

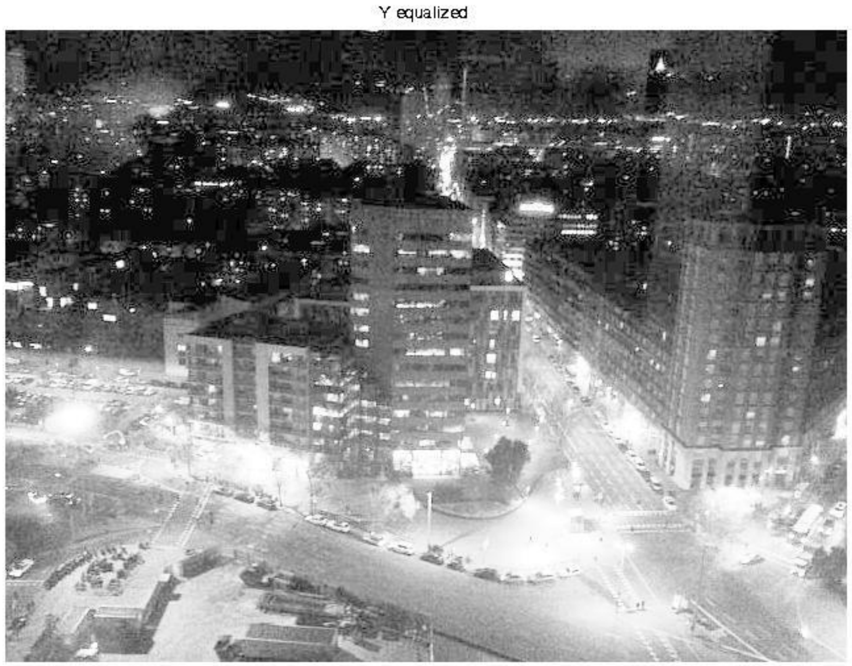
- The most frequently adopted directives
- Some directives suitable for arrays

The most frequently adopted directives

The application: image histogram equalization

- MATLAB: `y_eq = histeq(inp_y, 255);`





- See Freescale' application note: `AN4318.pdf, rev. 0, Jun 2011`

We will see...

- **Step after step how the various directives affect performance**
  1. set_directive_loop_tripcount -min 640 -max 1920 func_name/Lx

     set_directive_loop_tripcount -min 480 -max 1080 func_name/Ly
     - To help the compiler optimize an unbounded loop
  2. set_directive_dataflow top_func_name
     - To make the 3 subroutines concurrent (improves latency and throughput)
  3. config_dataflow -default_channel fifo -fifo_depth 2
     - To change the default interface of dataflow from ping-pong RAM to FIFO (all in all video data are streamed) buffering
  4. set_directive_pipeline
     - To pipeline the various routines (improves latency and throughput)
  5. set_directive_dependence false

     set_directive_resource -core RAM_T2P_BRAM array_name
     - To improve throughput in an array mapped as True dual-Port BRAM

The source code: my_defines.h

```
#define MAX_WIDTH          1080
#define MAX_HEIGHT         1920

typedef struct {
  unsigned char R;
  unsigned char G;
  unsigned char B;
  } RGB_t;

RGB_t inp_img[MAX_HEIGHT][MAX_WIDTH];
```

The source code: top_img_hist_equaliz1() Design Under Test

```
void top_img_hist_equaliz1( uint11 width, uint11 height,
                            uint25 cdf[GRAY_LEVELS], uint19 hist[GRAY_LEVELS],
                             RGB_t *inp_img,  RGB_t *out_img)
{
  #pragma HLS DATA_PACK variable=out_img
  #pragma HLS INTERFACE ap_fifo port=out_img
  #pragma HLS INTERFACE ap_fifo port=inp_img
  #pragma HLS DATA_PACK variable=inp_img

  /* NOTE: in HW this is a big chunk of memory */
  RGB_t yuv[MAX_WIDTH*MAX_HEIGHT];
  RGB_t yeq[MAX_WIDTH*MAX_HEIGHT];

  // RGB to YUV conversion of current image I(n)
  rgb2yuv(width, height, inp_img, yuv);
  // equalize the current image I(n) and compute its histogram for next image I(n+1)
  img_hist_equaliz1(width, height, hist, cdf, yuv, yeq);
  // YUV to RGB conversion of current image I(n)
  yuv2rgb(width, height, yeq, out_img);

}
```

The source code: rgb2yuv()

```
void rgb2yuv (uint11 width, uint11 height, RGB_t *in, RGB_t *out) {
  uint11 x, y;
  uint8 R, G, B, Y, U, V;
  const int9 Wrgb[3][3] = { {66,129,25}, {-38,-74,112}, {122,-94,-18} };

Ly: for (y=0; y<height; y++) {
     Lx: for (x=0; x<width; x++) {

      R = in[y*MAX_WIDTH +x].R;
      G = in[y*MAX_WIDTH +x].G;
      B = in[y*MAX_WIDTH +x].B;

      Y = ( (Wrgb[0][0]*R + Wrgb[0][1]*G + Wrgb[0][2]*B + 128) >> 8) +  16;
      U = ( (Wrgb[1][0]*R + Wrgb[1][1]*G + Wrgb[1][2]*B + 128) >> 8) + 128;
      V = ( (Wrgb[2][0]*R + Wrgb[2][1]*G + Wrgb[2][2]*B + 128) >> 8) + 128;

      out[y*MAX_WIDTH +x].R = Y;
      out[y*MAX_WIDTH +x].G = U;
      out[y*MAX_WIDTH +x].B = V;
      }
   }
}
```

The source code: yuv2rgb()

```
void yuv2rgb( uint11 width, uint11 height, RGB_t *inp_img, RGB_t *out_img)
 {
   int i; uint11  x, y; uint8   R, G, B, Y, U, V; int9    C, D, E;
   const int11 Wyuv[3][3] = { {298, 0, 409}, {298, -100, -208}, {298,  516, 0}  };

  Ly: for (y=0; y<height; y++) {
    Lx: for (x=0; x<width; x++) {
    Y=inp_img[y*MAX_WIDTH +x].R; U=inp_img[y*MAX_WIDTH +x].G; V=inp_img[y*MAX_WIDTH +x].B;

    C = Y - 16; D = U - 128; E = V - 128;
    R = CLIP(( Wyuv[0][0] * C                + Wyuv[0][2] * E + 128) >> 8);
    G = CLIP(( Wyuv[1][0] * C + Wyuv[1][1] * D + Wyuv[1][2] * E + 128) >> 8);
    B = CLIP(( Wyuv[2][0] * C + Wyuv[2][1] * D                + 128) >> 8);

    out_img[y*MAX_WIDTH +x].R = R;
    out_img[y*MAX_WIDTH +x].G = G;
    out_img[y*MAX_WIDTH +x].B = B;

    }
  }
  return;
}
```

The source code: compute_cdf()

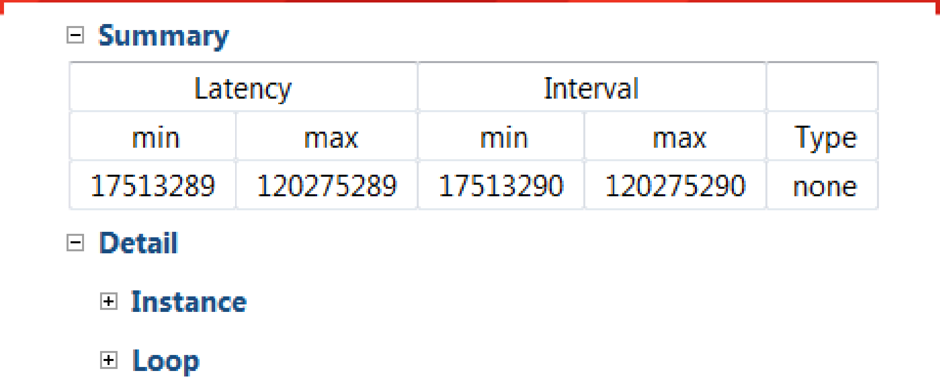
```
void compute_cdf(uint19 hist[GRAY_LEVELS], uint25 cdf[GRAY_LEVELS])
{
   uint8 i, k;

CDF_L1:    for (k = 0; k < GRAY_LEVELS-1; k++)
   {
       uint25 tmp_cdf=0;
           CDF_L2: for (i = 0; i<=k; i++)
           {
                     tmp_cdf = tmp_cdf + hist[i];
           }
           cdf[k] = tmp_cdf;
   }

   return;

}
```

The source code: img_hist_equaliz1

```
void img_hist_equaliz1( uint11 width, uint11 height, uint19 hist[GRAY_LEVELS],
                        uint25 cdf[GRAY_LEVELS], RGB_t *inp_img, RGB_t *out_img)
{
   int i; uint11 x, y; uint8 Y, U, V, cdf_temp; uint32  cdf_mult, gain;
   int numerator = 256*1024*(GRAY_LEVELS-1);
   int denominator = (height*width-1);
   gain =numerator / denominator;
   uint8 old=0, val; uint32 acc=0;
   Ly: for (y=0; y<height; y++) {
     Lx: for (x=0; x<width; x++) {
          Y = inp_img[y*MAX_WIDTH+x].R; U = inp_img[y*MAX_WIDTH+x].G; V = inp_img[y*MAX_WIDTH+x].B;
          //hist[Y]=hist[Y]+1;
          val = Y; if (old==val) acc++; else { hist[old]=acc; acc=hist[val]+1;} old = val;
          cdf_mult = (cdf[Y]-1) * gain ; // histogram equalization
          if (cdf_mult >= (GRAY_LEVELS-1)) cdf_temp = (GRAY_LEVELS-1);
          if (cdf_mult <= 0 )      cdf_temp = 0;
          else                                cdf_temp = (uint8) cdf_mult;
          Y = cdf_temp;
          out_img[y*MAX_WIDTH+x].R = Y; out_img[y*MAX_WIDTH+x].G = U;  out_img[y*MAX_WIDTH+x].B = V;
     } // end of Lx loop
   } // end of Ly loop
   hist[old]=acc;
}
```

Solution1: synthesis estimation

- **set_directive_loop_tripcount -min 640 -max 1920 func_name/Lx**
- **set_directive_loop_tripcount -min 480 -max 1080 func_name/Ly**



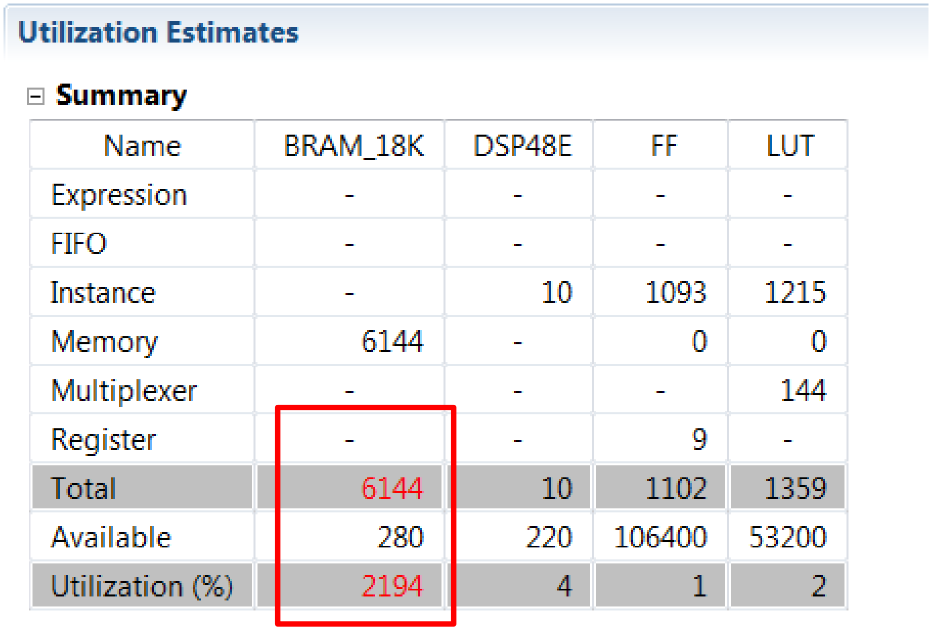
Summary

Latency		Interval		
min	max	min	max	Type
17513289	120275289	17513290	120275290	none

Detail

Instance

Loop



Utilization Estimates

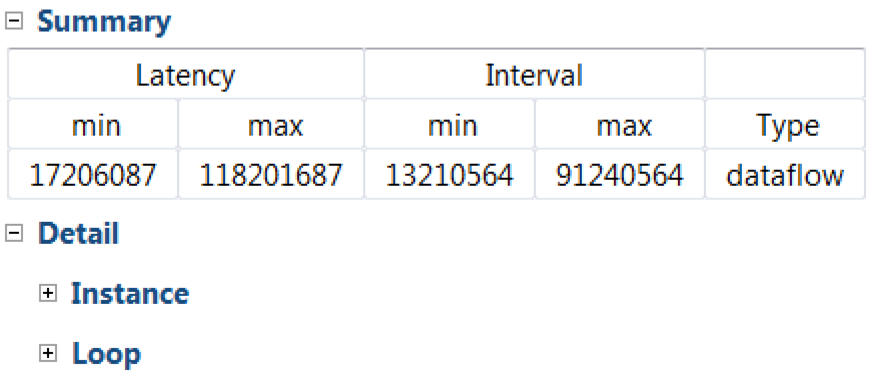
Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	10	1093	1215
Memory	6144	-	0	0
Multiplexer	-	-	-	144
Register	-	-	9	-
Total	6144	10	1102	1359
Available	280	220	106400	53200
Utilization (%)	2194	4	1	2

Look at how many BRAMs!

Solution2: synthesis estimation

- As solution1 plus **set_directive_dataflow**



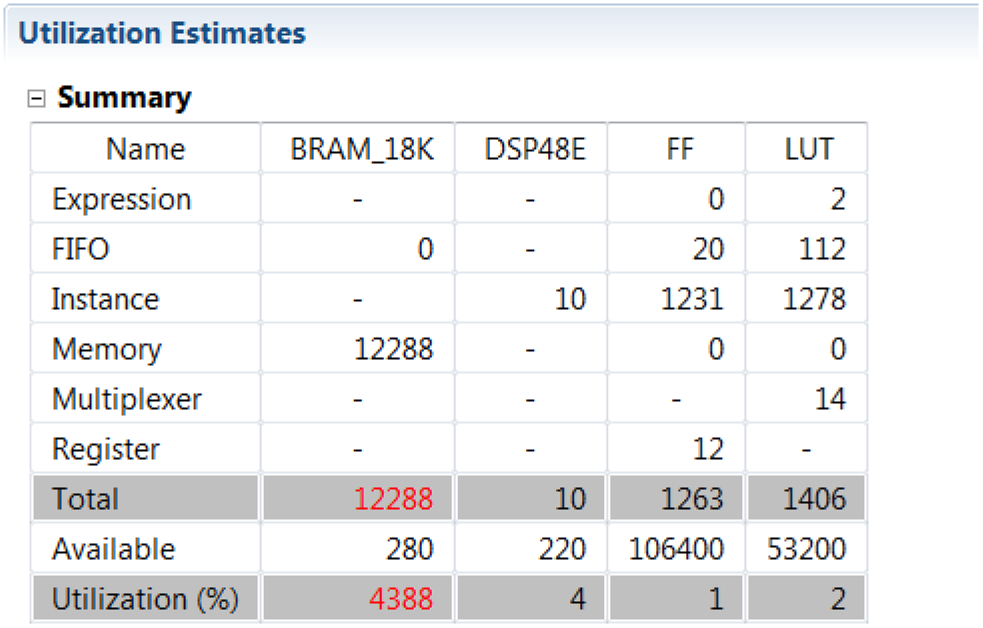
⊟ **Summary**

Latency		Interval		
min	max	min	max	Type
17206087	118201687	13210564	91240564	dataflow

⊟ **Detail**

⊞ **Instance**

⊞ **Loop**



Utilization Estimates

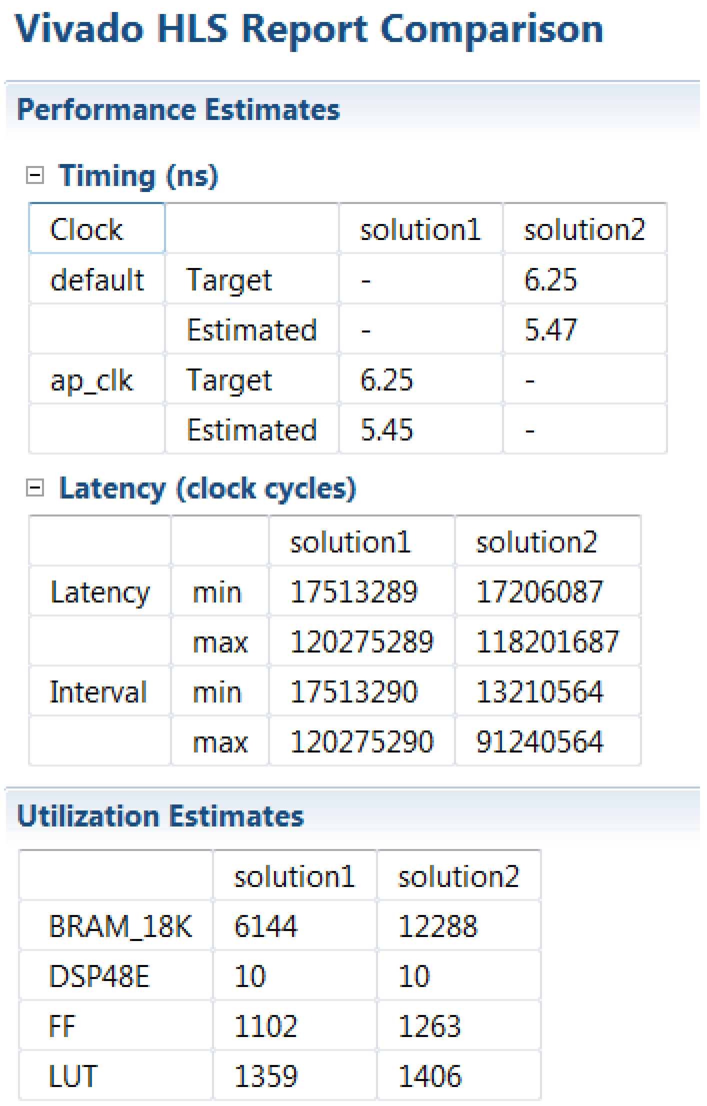
⊟ **Summary**

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	2
FIFO	0	-	20	112
Instance	-	10	1231	1278
Memory	12288	-	0	0
Multiplexer	-	-	-	14
Register	-	-	12	-
Total	12288	10	1263	1406
Available	280	220	106400	53200
Utilization (%)	4388	4	1	2

solution2 vs. solution1

Worst performance

due to double-buffers:



Vivado HLS Report Comparison

Performance Estimates

⊟ Timing (ns)

Clock		solution1	solution2
default	Target	-	6.25
	Estimated	-	5.47
ap_clk	Target	6.25	-
	Estimated	5.45	-

⊟ Latency (clock cycles)

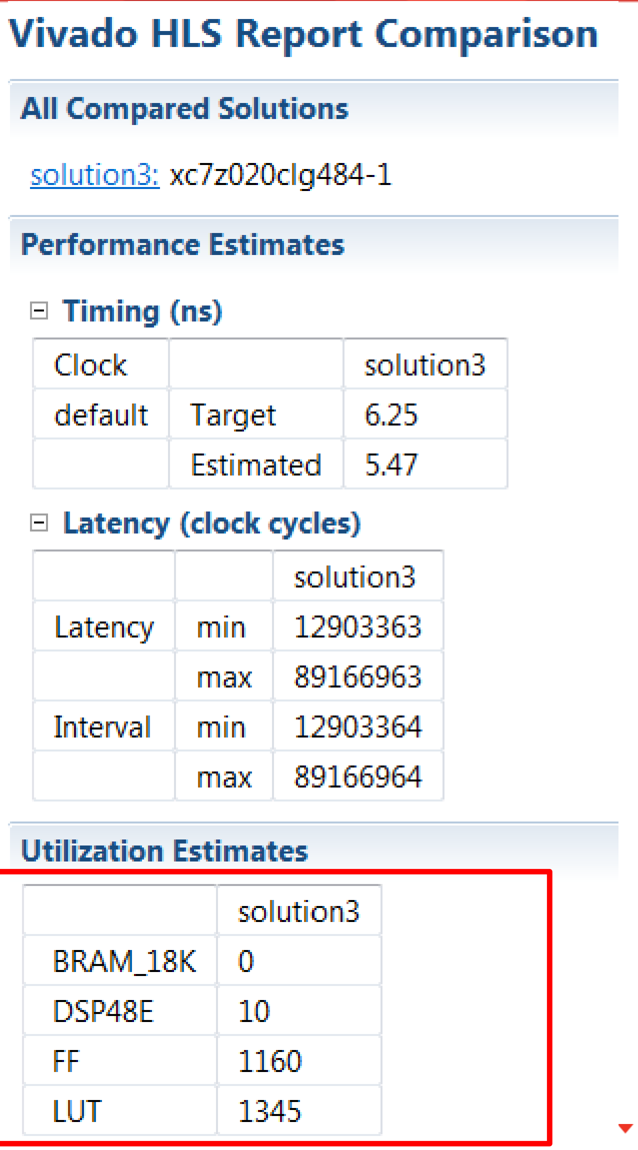
		solution1	solution2
Latency	min	17513289	17206087
	max	120275289	118201687
Interval	min	17513290	13210564
	max	120275290	91240564

Utilization Estimates

	solution1	solution2
BRAM_18K	6144	12288
DSP48E	10	10
FF	1102	1263
LUT	1359	1406

Solution3: synthesis estimation

- As solution2 plus **config_dataflow -default_channel fifo -fifo_depth 2** in the script.tcl constraints file



Vivado HLS Report Comparison

All Compared Solutions

solution3: xc7z020clg484-1

Performance Estimates

Timing (ns)

Clock		solution3
default	Target	6.25
	Estimated	5.47

Latency (clock cycles)

		solution3
Latency	min	12903363
	max	89166963
Interval	min	12903364
	max	89166964

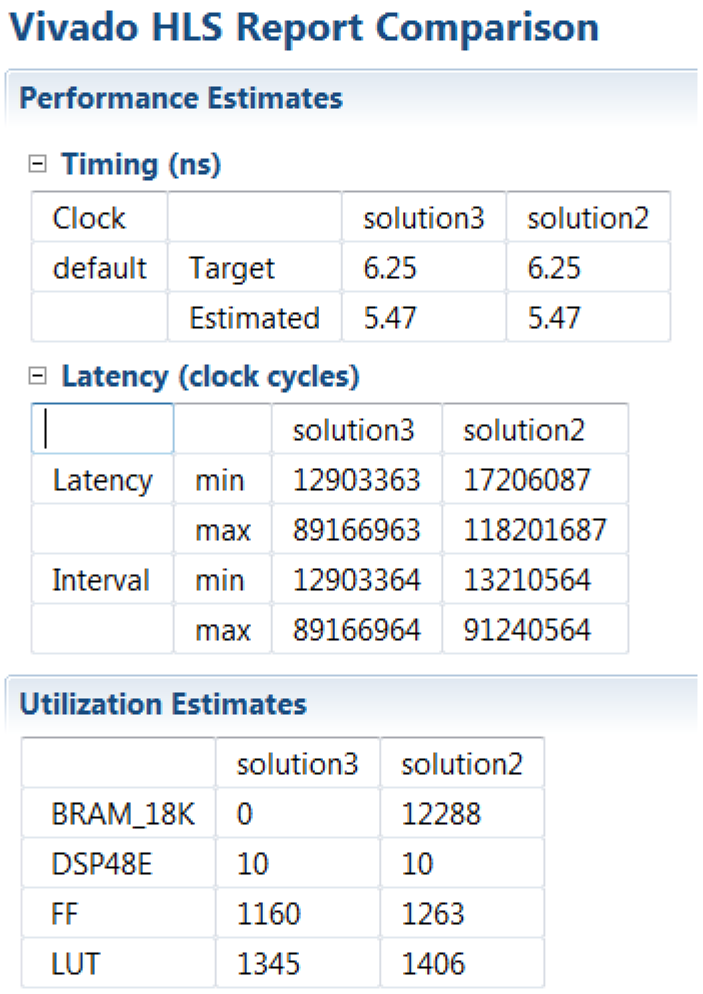
Utilization Estimates

	solution3
BRAM_18K	0
DSP48E	10
FF	1160
LUT	1345

BRAM disappeared!

Solution3 vs. solution2

- **Small FIFO are much better than double buffers**



Vivado HLS Report Comparison

Performance Estimates

Timing (ns)

Clock		solution3	solution2
default	Target	6.25	6.25
	Estimated	5.47	5.47

Latency (clock cycles)

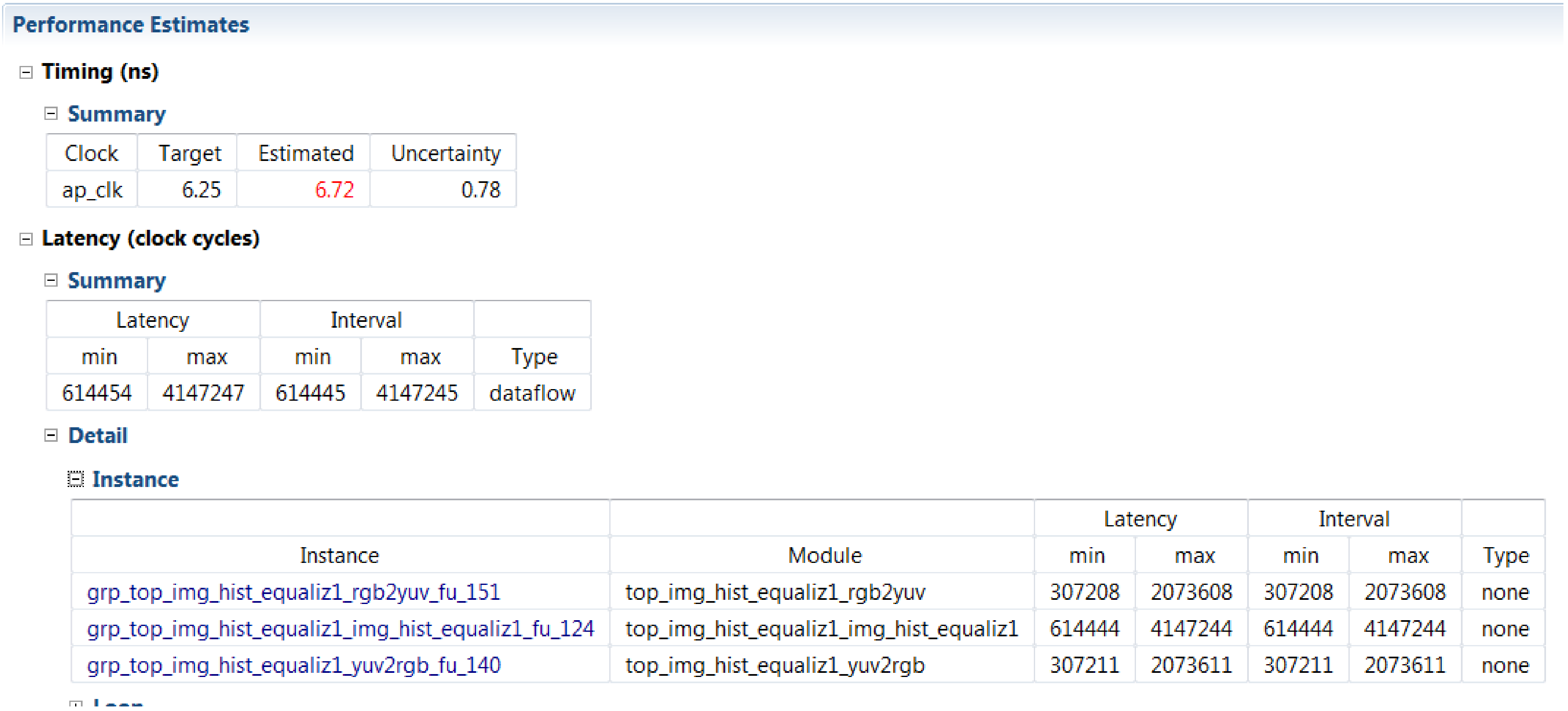
		solution3	solution2
Latency	min	12903363	17206087
	max	89166963	118201687
Interval	min	12903364	13210564
	max	89166964	91240564

Utilization Estimates

	solution3	solution2
BRAM_18K	0	12288
DSP48E	10	10
FF	1160	1263
LUT	1345	1406

Solution4: synthesis estimation

- As solution3 plus **set_directive_pipeline** for all Lx loops



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	6.25	6.72	0.78

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
614454	4147247	614445	4147245	dataflow

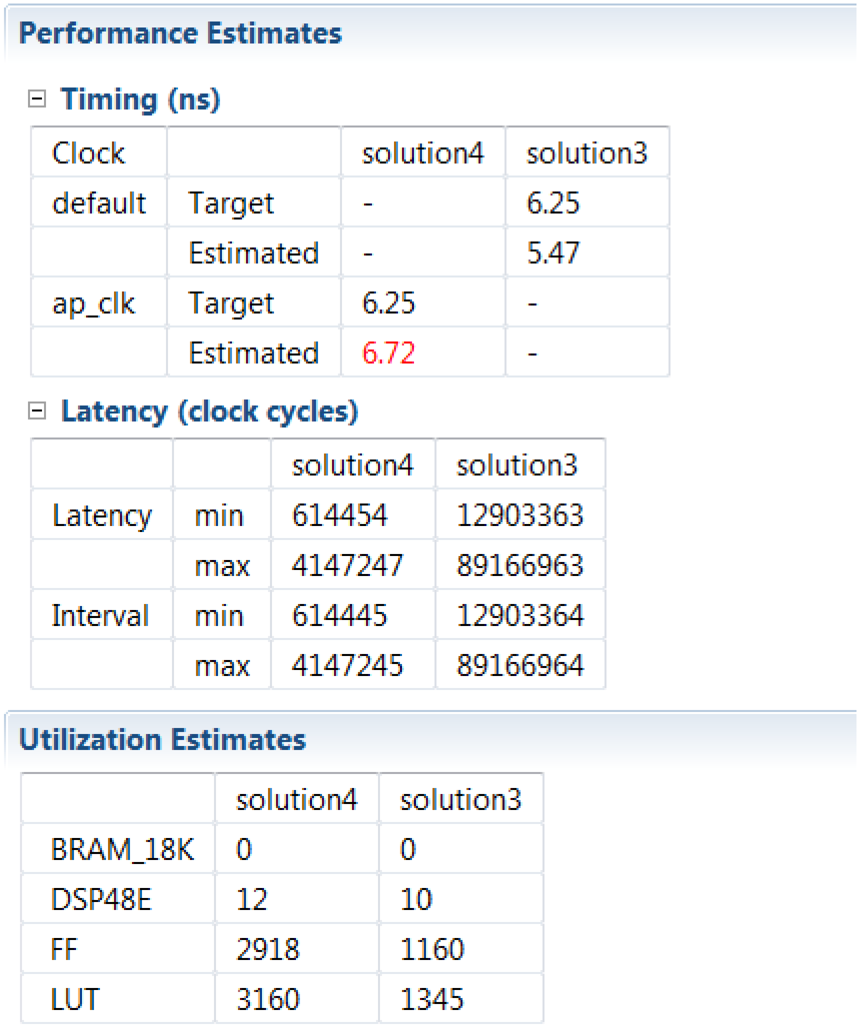
⊟ Detail

⊟ Instance

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_top_img_hist_equaliz1_rgb2yuv_fu_151	top_img_hist_equaliz1_rgb2yuv	307208	2073608	307208	2073608	none
grp_top_img_hist_equaliz1_img_hist_equaliz1_fu_124	top_img_hist_equaliz1_img_hist_equaliz1	614444	4147244	614444	4147244	none
grp_top_img_hist_equaliz1_yuv2rgb_fu_140	top_img_hist_equaliz1_yuv2rgb	307211	2073611	307211	2073611	none

solution4 vs. solution3

- **Better throughput and latency**



Performance Estimates

⊟ **Timing (ns)**

Clock		solution4	solution3
default	Target	-	6.25
	Estimated	-	5.47
ap_clk	Target	6.25	-
	Estimated	6.72	-

⊟ **Latency (clock cycles)**

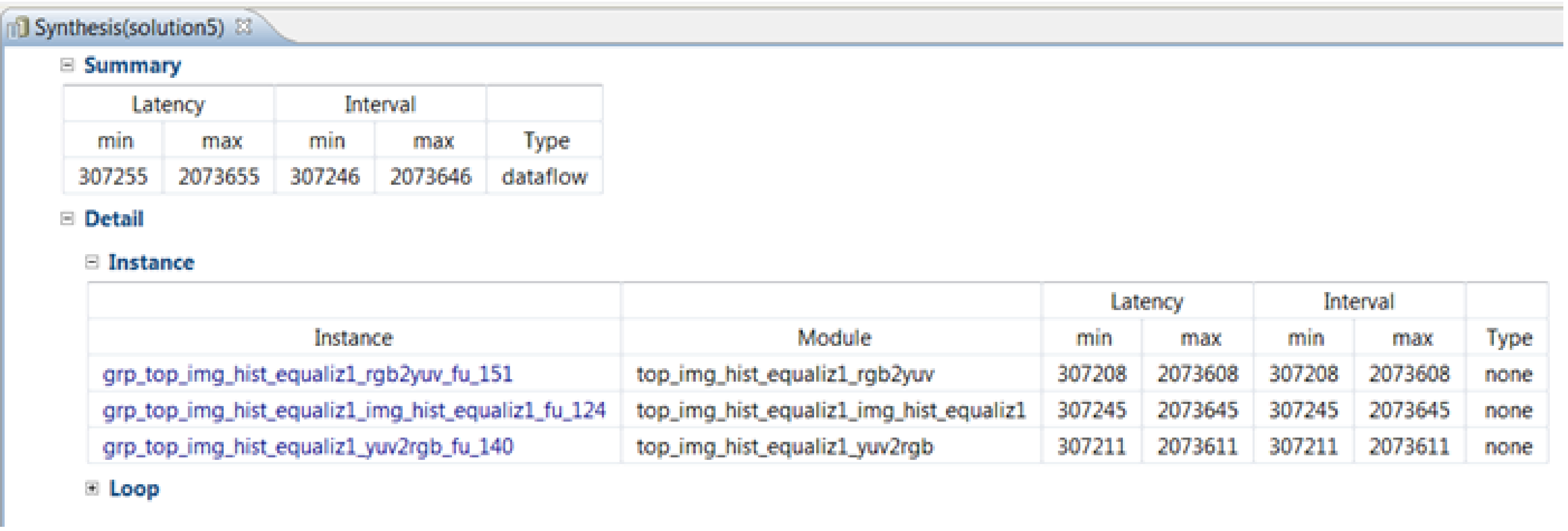
		solution4	solution3
Latency	min	614454	12903363
	max	4147247	89166963
Interval	min	614445	12903364
	max	4147245	89166964

Utilization Estimates

	solution4	solution3
BRAM_18K	0	0
DSP48E	12	10
FF	2918	1160
LUT	3160	1345

Solution5

- **As solution4 plus**
  - set_directive_resource -core RAM_T2P_BRAM "dpram_array_hist" hist
  - set_directive_dependence -variable hist -type inter -dependent false "dpram_array_hist"



Synthesis(solution5)

Summary

Latency		Interval		
min	max	min	max	Type
307255	2073655	307246	2073646	dataflow

Detail

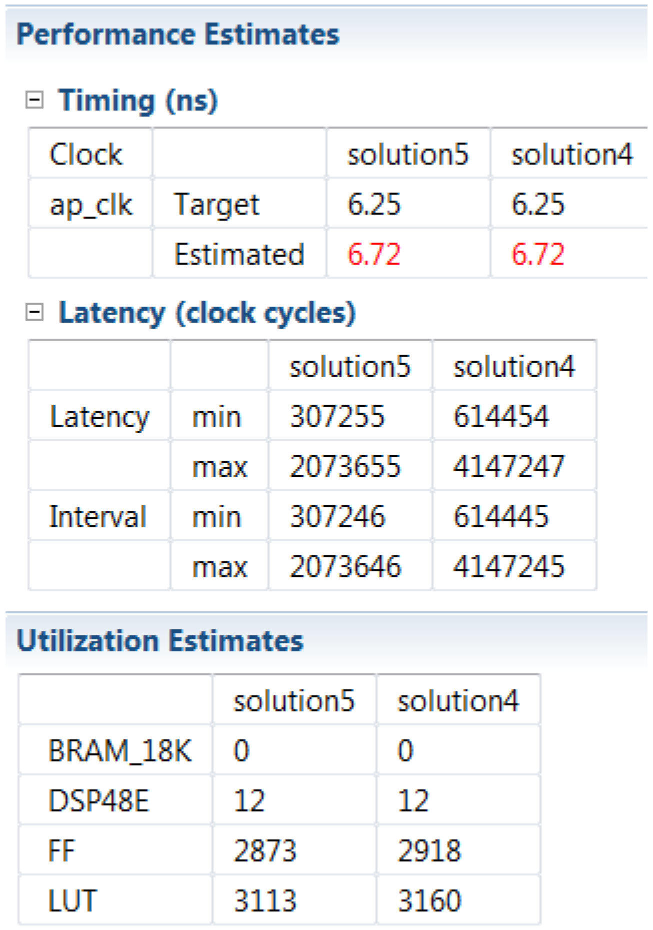
Instance

		Latency		Interval		
Instance	Module	min	max	min	max	Type
grp_top_img_hist_equaliz1_rgb2yuv_fu_151	top_img_hist_equaliz1_rgb2yuv	307208	2073608	307208	2073608	none
grp_top_img_hist_equaliz1_img_hist_equaliz1_fu_124	top_img_hist_equaliz1_img_hist_equaliz1	307245	2073645	307245	2073645	none
grp_top_img_hist_equaliz1_yuv2rgb_fu_140	top_img_hist_equaliz1_yuv2rgb	307211	2073611	307211	2073611	none

Loop

Solution5 vs solution4

- Better throughput and latency



Performance Estimates

Timing (ns)

Clock		solution5	solution4
ap_clk	Target	6.25	6.25
	Estimated	6.72	6.72

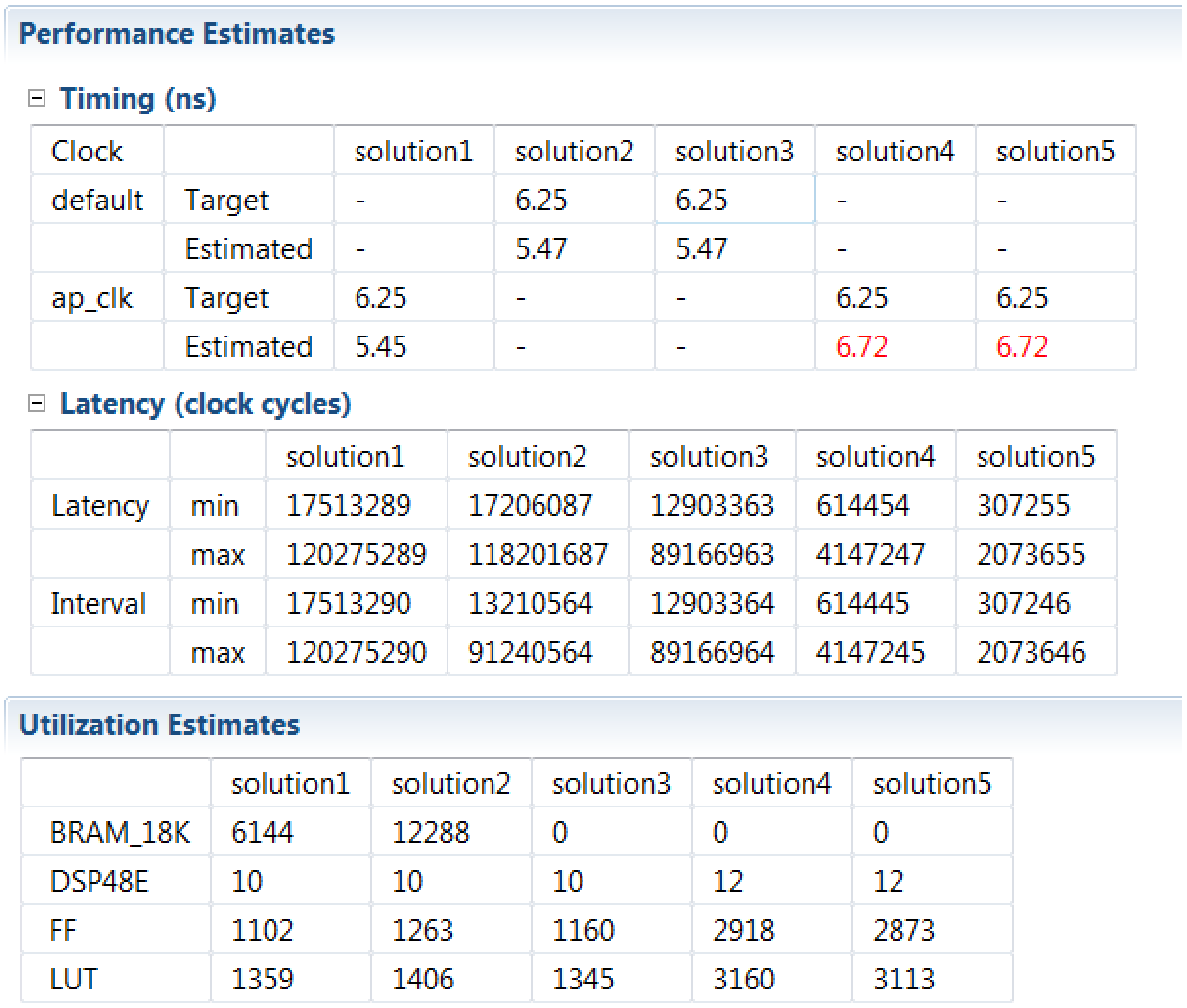
Latency (clock cycles)

		solution5	solution4
Latency	min	307255	614454
	max	2073655	4147247
Interval	min	307246	614445
	max	2073646	4147245

Utilization Estimates

	solution5	solution4
BRAM_18K	0	0
DSP48E	12	12
FF	2873	2918
LUT	3113	3160

Summary



Performance Estimates

Timing (ns)

Clock		solution1	solution2	solution3	solution4	solution5
default	Target	-	6.25	6.25	-	-
	Estimated	-	5.47	5.47	-	-
ap_clk	Target	6.25	-	-	6.25	6.25
	Estimated	5.45	-	-	6.72	6.72

Latency (clock cycles)

		solution1	solution2	solution3	solution4	solution5
Latency	min	17513289	17206087	12903363	614454	307255
	max	120275289	118201687	89166963	4147247	2073655
Interval	min	17513290	13210564	12903364	614445	307246
	max	120275290	91240564	89166964	4147245	2073646

Utilization Estimates

	solution1	solution2	solution3	solution4	solution5
BRAM_18K	6144	12288	0	0	0
DSP48E	10	10	10	12	12
FF	1102	1263	1160	2918	2873
LUT	1359	1406	1345	3160	3113

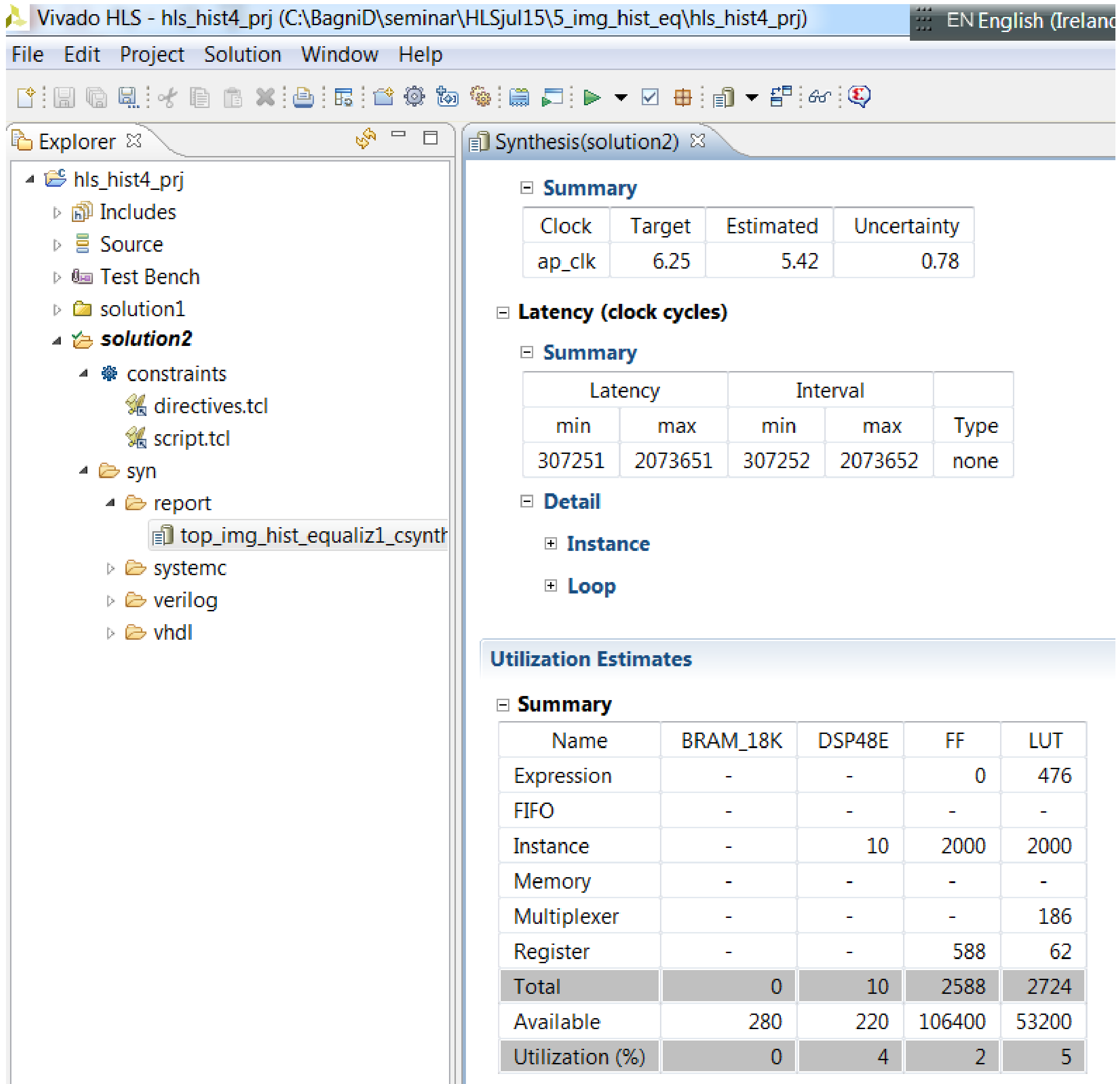
A new approach

- Since the 3 functions have the same double FOR loop....
- ...by manually merging all 3 loops into a single one we should save resources and get a better performance

Exercise on image Histogram equalization

- IN ORDER 1) TO DECREASE RESOURCES AND 2) TO AVOID USING DATA FLOW, WE CAN MANUALLY MERGE (INLINE) THE 3 INNER FUNCTIONS: rgb2yuv(), img_hist_equaliz1(), yuv2rgb() INTO THE TOP LEVEL top_img_hist_equaliz1
- YOU NEED ONLY 2 LOOPS: Ly and Lx
- Steps:
- 1) Copy src3 directory into src4 new directory
- 2) edit the files into src4 to manually merge the three functions
- 3) create a new HLS project: targeting again ZC702 160MHz
- 4) optimize with PIPELINE and DEPENDENCE as you did for src3
- 5) compares best results of the two different HLS projects

Synthesis estimation of new approach


Vivado HLS - hls_hist4_prj (C:\BagniD\seminar\HLSjul15\5_img_hist_eq\hls_hist4_prj)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	6.25	5.42	0.78

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
307251	2073651	307252	2073652	none

Detail

Instance

Loop

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
Expression	-	-	0	476
FIFO	-	-	-	-
Instance	-	10	2000	2000
Memory	-	-	-	-
Multiplexer	-	-	-	186
Register	-	-	588	62
Total	0	10	2588	2724
Available	280	220	106400	53200
Utilization (%)	0	4	2	5

XILINX

Synthesis performance summary (top*equaliz1 & top*equaliz2)

version		FF	LUT	BRAM	DSP48	worst latency		directives
orig. code: top_img_hist_equaliz1								
solution1		1102	1359	6144	10	120275290		set the min and max tripcount for all the loops of all funcs
solution2		1263	1406	12288	10	91240564		as solution 1, plus set_directive_dataflow for top_img_hist_equaliz1()
solution3		1160	1345	0	10	89166964		as solution 2, plus config_dataflow -default_channel fifo -fifo_depth 2
solution4		2918	3160	0	12	4147245		as solution3, plus set_directive_pipeline for all Lx loops of all funcs
solution5		2873	3133	0	12	2073646		as solution4, plus set_directive_dependence for dpram_array_hist()
new code: top_img_hist_equaliz2								
solution		2588	2724	0	10	2073651		all above directives

Better optimization for CDF(): original

```
#define OT_OPTIMIZED

#ifndef OT_OPTIMIZED
void compute_cdf(uint19 hist[GRAY_LEVELS], uint25 cdf[GRAY_LEVELS])
{
   uint8 i, k;
   CDF_L1:for (k = 0; k < GRAY_LEVELS-1; k++) {
      uint25 tmp_cdf=0;
      CDF_L2: for (i = 0; i<=k; i++) {
         #pragma HLS loop_tripcount min=1 max=256 avg=256
         // cdf for current picture from past picture histogram
         tmp_cdf = tmp_cdf + hist[i];
      }
      cdf[k] = tmp_cdf;
   }
   return;
 }
```

Better optimization for CDF(): optimized

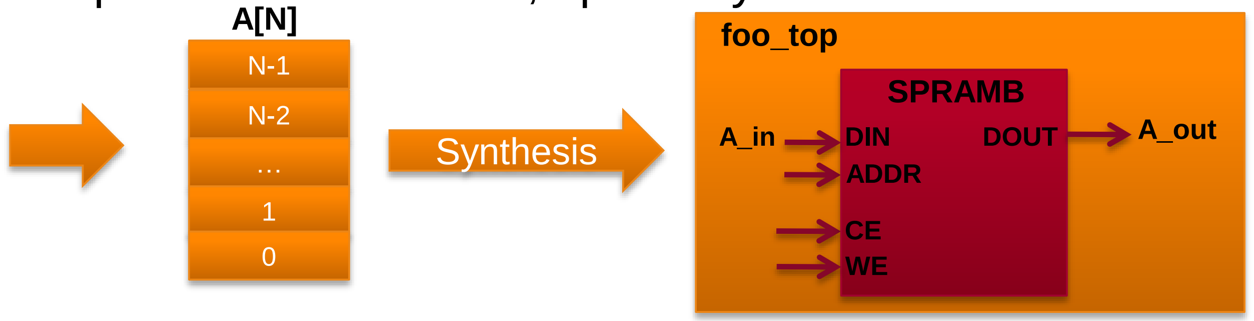
```
#else
void compute_cdf(uint19 hist[GRAY_LEVELS], uint25 cdf[GRAY_LEVELS])
{
   uint8  k;
   uint25 tmp_cdf=0;

   CDF_L1:for (k = 0; k < GRAY_LEVELS-1; k++)
   {
         #pragma HLS pipeline II=1
         // cdf for current picture from past picture histogram
         tmp_cdf = tmp_cdf + hist[k];
         cdf[k] = tmp_cdf;
   }
   return;

}
#endif
```

Some directives suitable for arrays

Review: Arrays in HLS

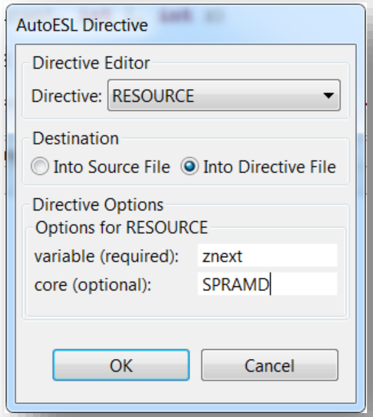
- **An array in C code is implemented by a memory in the RTL**
  - By default, arrays are implemented as RAMs, optionally a FIFO

```
void foo_top(int x, …){
  int A[N];
  L1: for (i = 0; i < N; i++)
       A[i+x] = A[i] + i;
}
```



- **The array can be targeted to any memory resource in the library**
  - The ports and sequential operation are defined by the library model
    - All RAMs are listed in the AutoESL Library Guide

Example: Array "znext" is targeted to a single port distributed RAM resource


AutoESL Directive

Directive Editor

Directive: RESOURCE

Destination

Into Source File [✓ Into Directive File]

Directive Options

Options for RESOURCE

variable (required): znext

core (optional): SPRAMD

Vertical Array Mapping

- **In vertical mapping, arrays with same length are concatenated by to produce an array with larger bit-widths.**
  - i.e. we could move from 2 BRAM units of 1K (depth) x 18bit words each one to only 1 BRAM unit of 1K (depth) x 36bit

- **We have seen the effect in the image histogram application:**

```
– set_directive_array_map -instance name -mode vertical “func_name" array1_name
– set_directive_array_map -instance name -mode vertical “func_name" array2_name
```

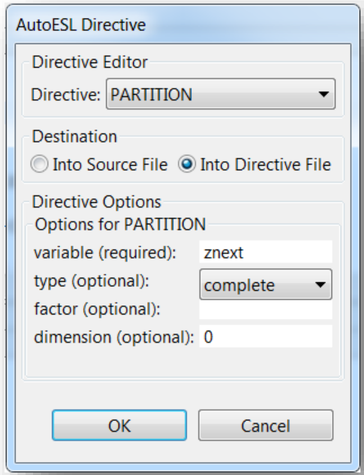
Horizontal Array Mapping

- **In horizontal mapping, small arrays are concatenated one after the other to produce a longer array with same bit-widths.**
  - `set_directive_array_map -instance name -mode horizontal "func_name" array1_name`
  - `set_directive_array_map -instance name -mode horizontal "func_name" array2_name`

- **Although horizontal mapping can result in using less RAM components and hence improve area, it can have an impact on throughput and performance**

Array Partitioning

▪ Partitioning breaks an array into smaller elements


AutoESL Directive

Directive Editor

Directive: PARTITION

Destination

( ) Into Source File (●) Into Directive File

Directive Options

Options for PARTITION

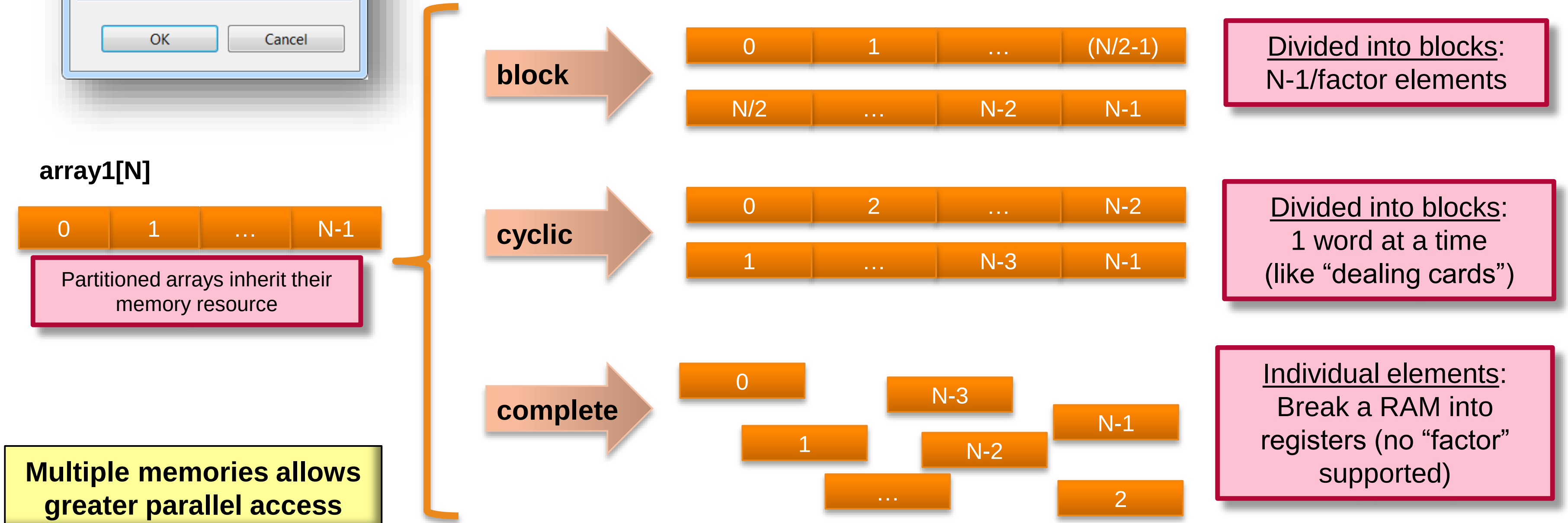
variable (required): znext

type (optional): complete

factor (optional):

dimension (optional): 0

- Arrays can be split along any dimension
  - If none is specified dimension zero is assumed
    - Dimension zero means all dimensions



Partitioned arrays inherit their memory resource

Divided into blocks: N-1/factor elements

Divided into blocks: 1 word at a time (like "dealing cards")

Individual elements: Break a RAM into registers (no "factor" supported)

Multiple memories allows greater parallel access