Article

Template Matching. Optimizing performance and area in Vivado HLS

Vlad-Eusebiu Baciu

Student ID: 0577953
E-mail: vlad-eusebiu.baciu@vub.be
Course: Multiprocessors and Reconfigurable Architectures

Abstract: The task consists to implement of a simple image detector technique that attempts to evaluate the matching of two images. A simple cross-correlation technique is proposed: sum of absolute differences (SAD). It uses an image patch (template), tailored to a specific feature of the search image. In order to apply this template matching method the image is normalized and grayscaled. Two designs with optimizations targeting performance and area have been created in Vivado HLS.

Keywords: template matching; image detector; VIVADO HLS

0. Introduction

The algorithm is able to receive two stream of pixels from the original image and template image in a BMP format (RGB, 3 bytes/pixel). On both input images two algorithms are applied: grayscaling and normalization. The algorithm returns a new image in which the original input template image is superimposed over the rest of the grayscale image. The grayscale algorithm supports three standards: LUMA YUV, ITU-R BT.709 and ITU-R BT.2100. The top function calls the grayscale function seting the enconding type as LUMA YUV and the normalization function with a minumum value of 0 and a maximum value of 255 as parameters. These parameters can be configured by the user if access to top function is permitted and can not be modified from the test bench.

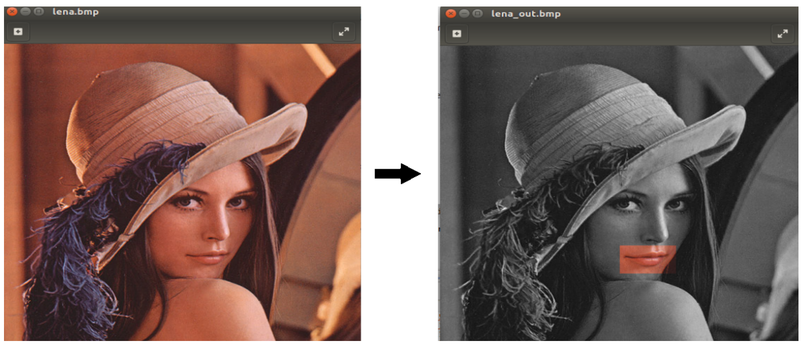


Figure 1. Template matching result: (**a**) Input image (**b**) Output image

1. Vivado HLS - optimization

Three Vivado HLS solution are presented in this report:

- Solution 1: default solution, no optimizations are applied except HLS loop trip-count;
- Solution 2: few optimization directives applied in order to achive the highest performance;
- Solution 3: few optimization directives applied in order to reduce area and resource consumption.

Vivado HLS provides pragmas that can be used to optimize the design: reduce latency, improve throughput performance, and reduce area and device resource utilization of the resulting RTL code. These pragmas can be added directly to the source code for the kernel. The following table contains the pragma directives used in this project.

Table 1. Vivado HLS pragmas used in the project.

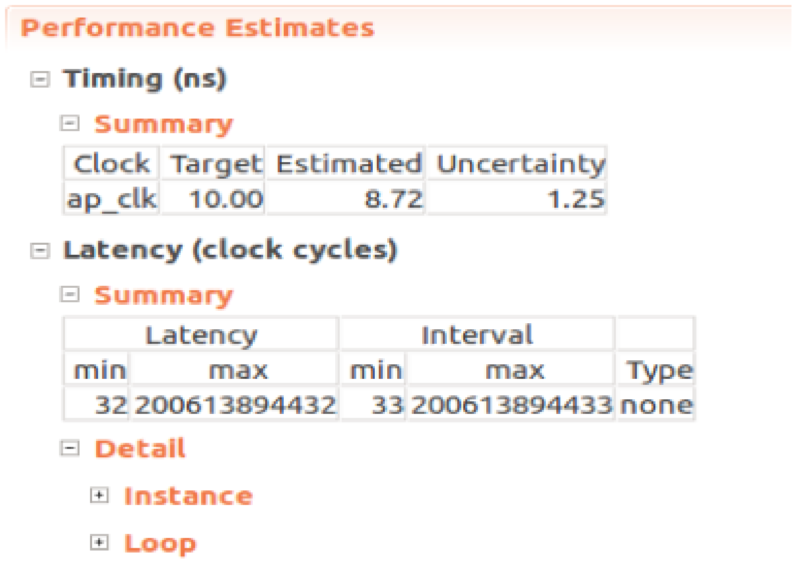
Pragma	Optimization type
LOOP TRIPCOUNT	performance
ARRAY PARTITION	performance
LOOP UNROLL	performance
DATAFLOW	performance
STREAM	performance
ARRAY MAP	area
INLINE	area
BIT-ACCURATE TYPES	area
ALLOCATION	area

LOOP TRIPCOUNT: Can be applied to a loop to manually specify the total number of iterations performed by a loop.
ARRAY PARTITION: Partitions an array into smaller arrays or individual elements and increases the amount of read and write ports for the storage.
LOOP UNROLL: Transforms loops by creating multiples copies of the loop body in the RTL design, which allows some or all loop iterations to occur in parallel.
DATAFLOW: Enables task-level pipelining, allowing functions and loops to overlap in their operation, increasing the concurrency of the RTL implementation.
STREAM: Top-level function array parameters are implemented as a RAM interface port.
ARRAY MAP: Combines multiple smaller arrays into a single large array to help reduce block RAM resources.
INLINE: Allows operations within the function to be shared and optimized more effectively with surrounding operations.
BIT-ACCURATE TYPES: Allow any arbitrary bit width to be specified.
ALLOCATION: Limits resource allocation in the implemented kernel.

2. Results

2.1. Default solution

The default solution is presented below in which no optimization is used. Vivado HLS reports the total latency of each loop, which is the number of clock cycles to execute all iterations of the loop. The loop latency is therefore a function of the number of loop iterations, or tripcount. Since the tripcount is a constant value in my case the **pragma loop tripcount** has been used for each loop.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.72	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
32	200613894432	33	200613894433	none

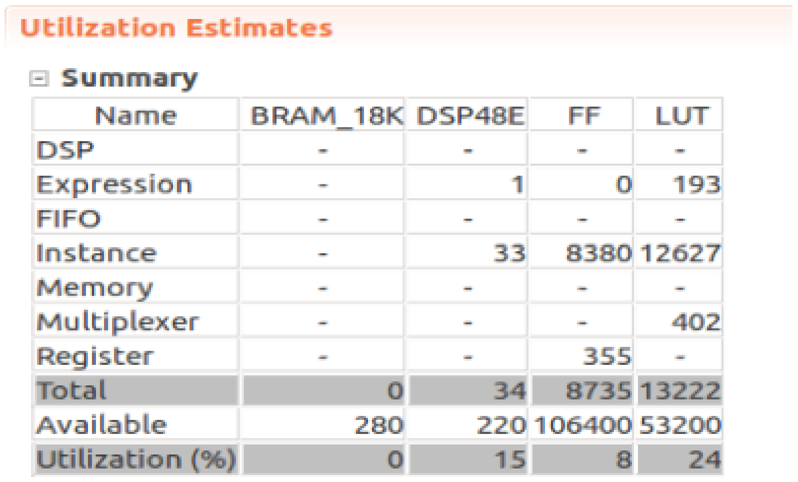
Detail

Instance

Loop

Figure 2. Performance estimates for default solution. No optimization included.

Without optimization the solution is implemented using a total of 15.6 % of resources.



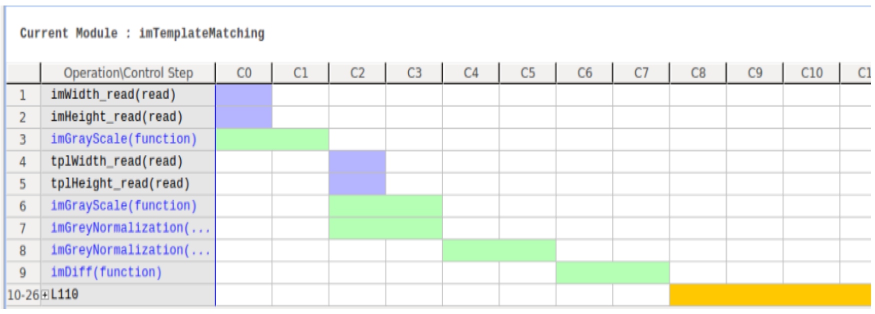
Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	1	0	193
FIFO	-	-	-	-
Instance	-	33	8380	12627
Memory	-	-	-	-
Multiplexer	-	-	-	402
Register	-	-	355	-
Total	0	34	8735	13222
Available	280	220	106400	53200
Utilization (%)	0	15	8	24

Figure 3. Utilization estimates for default solution. No optimization included.

In the following picture the operations and control steps are presented. The algorithm "executes" first the imGrayScale instance for the input image then in control step C2 starts executing the same instance for the template input image. At the same time the imGreyNormalization instance is "executed" for the previous image. In the control step C4 the template image, now in grayscale format, gets normalized. Starting with C6 the algorithm executes the template matching algorithm. Between C8 and C11 the output image is constructed as seen in Figure 1.



Current Module : imTemplateMatching

	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8	C9	C10	C1
1	imWidth_read(read)	🟦											
2	imHeight_read(read)	🟦											
3	imGrayScale(function)	🟩	🟩										
4	tplWidth_read(read)			🟦									
5	tplHeight_read(read)			🟦									
6	imGrayScale(function)			🟩	🟩								
7	imGreyNormalization(...			🟩	🟩								
8	imGreyNormalization(...					🟩	🟩						
9	imDiff(function)							🟩	🟩				
10-26	⊞L110									🟨	🟨	🟨	🟨

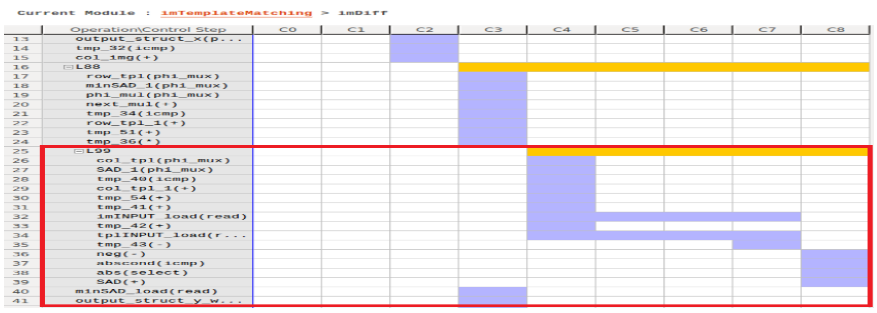
Figure 4. Performance analysis. No optimization included.

63 *2.2. Performance optimization*

64 The first design targets the performance. In order to achive the highest performance
65 the following methods have been used:

66 2.2.1. Loop unrolling

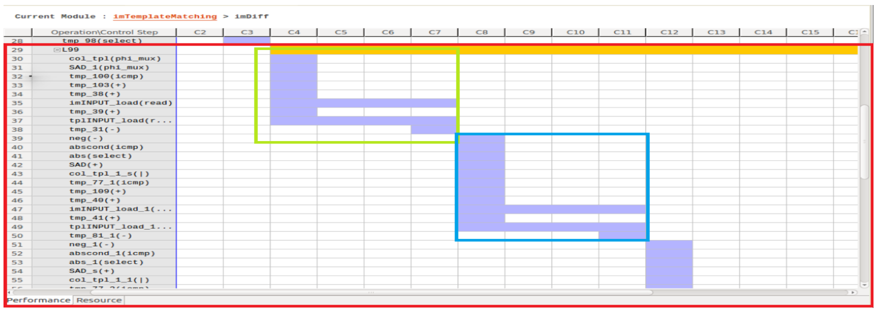
67 The UNROLL pragma transforms loops by creating multiples copies of the loop
68 body in the RTL design, which allows some or all loop iterations to occur in parallel. I
69 have chosen an unrolling factor of 4. Partially unrolling a loop lets you specify a factor
70 N, to create N copies of the loop body and reduce the loop iterations accordingly. The
71 impact of this method can be seen in Figure 6. Doing this the performance have been
72 increased with 14 % and the resource utilization reached 20 %.



Current Module : imTemplateMatching > imDiff

	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8
13	output_struct_x(p...			🟦						
14	tmp_32(icmp)			🟦						
15	col_img(+)			🟦						
16	⊟L88				🟨	🟨	🟨	🟨	🟨	🟨
17	row_tpl(phi_mux)				🟦					
18	minSAD_1(phi_mux)				🟦					
19	phi_mul(phi_mux)				🟦					
20	next_mul(+)				🟦					
21	tmp_34(icmp)				🟦					
22	row_tpl_1(+)				🟦					
23	tmp_51(+)				🟦					
24	tmp_36(*)				🟦					
25	⊟L99					🟨	🟨	🟨	🟨	🟨
26	col_tpl(phi_mux)					🟦				
27	SAD_1(phi_mux)					🟦				
28	tmp_40(icmp)					🟦				
29	col_tpl_1(+)					🟦				
30	tmp_54(+)					🟦				
31	tmp_41(+)					🟦				
32	imINPUT_load(read)					🟦	🟦	🟦	🟦	
33	tmp_42(+)					🟦				
34	tplINPUT_load(r...					🟦	🟦	🟦	🟦	
35	tmp_43(-)								🟦	
36	neg(-)									🟦
37	abscond(icmp)									🟦
38	abs(select)									🟦
39	SAD(+)									🟦
40	minSAD_load(read)				🟦					
41	output_struct_y_w...				🟦					

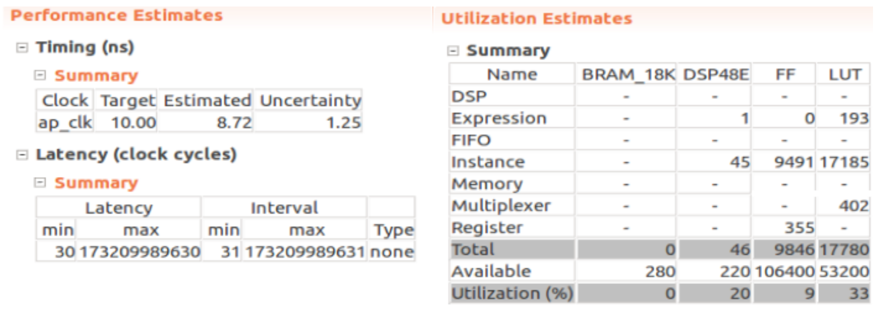
Figure 5. Loop labeled L99 after loop unrolling optimization.



Current Module : imTemplateMatching > imDiff

	Operation\Control Step	C2	C3	C4	C5	C6	C7	C8	C9	C10	C11	C12	C13	C14	C15	C1
28	tmp_98(select)		🟦													
29	⊟L99			🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨	🟨
30	col_tpl(phi_mux)			🟦												
31	SAD_1(phi_mux)			🟦												
32	tmp_100(icmp)			🟦												
33	tmp_103(+)			🟦												
34	tmp_38(+)			🟦												
35	imINPUT_load(read)			🟦	🟦	🟦	🟦									
36	tmp_39(+)			🟦												
37	tplINPUT_load(r...			🟦	🟦	🟦	🟦									
38	tmp_31(-)						🟦									
39	neg(-)							🟦								
40	abscond(icmp)							🟦								
41	abs(select)							🟦								
42	SAD(+)							🟦								
43	col_tpl_1_s(|)							🟦								
44	tmp_77_1(icmp)							🟦								
45	tmp_109(+)							🟦								
46	tmp_40(+)							🟦								
47	imINPUT_load_1(...							🟦	🟦	🟦	🟦					
48	tmp_41(+)							🟦								
49	tplINPUT_load_1...							🟦	🟦	🟦	🟦					
50	tmp_81_1(-)										🟦					
51	neg_1(-)											🟦				
52	abscond_1(icmp)											🟦				
53	abs_1(select)											🟦				
54	SAD_s(+)											🟦				
55	col_tpl_1_1(|)											🟦				

Figure 6. Loop labeled L99 before loop unrolling optimization. Unrolling factor 4.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.72	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
30	173209989630	31	173209989631	none

Utilization Estimates

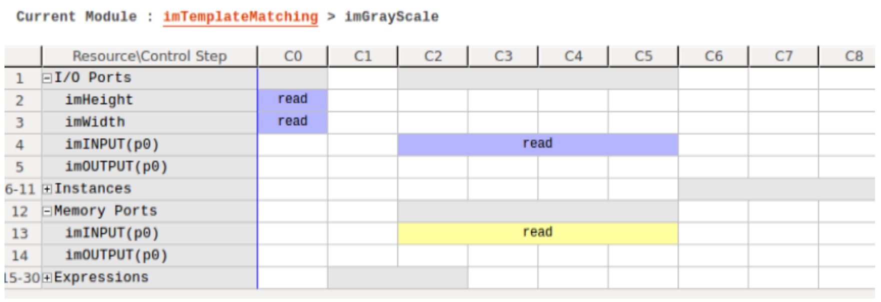
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	1	0	193
FIFO	-	-	-	-
Instance	-	45	9491	17185
Memory	-	-	-	-
Multiplexer	-	-	-	402
Register	-	-	355	-
Total	0	46	9846	17780
Available	280	220	106400	53200
Utilization (%)	0	20	9	33

Figure 7. Loop unrolling impact on performance and utilization.

2.2.2. Array partitioning

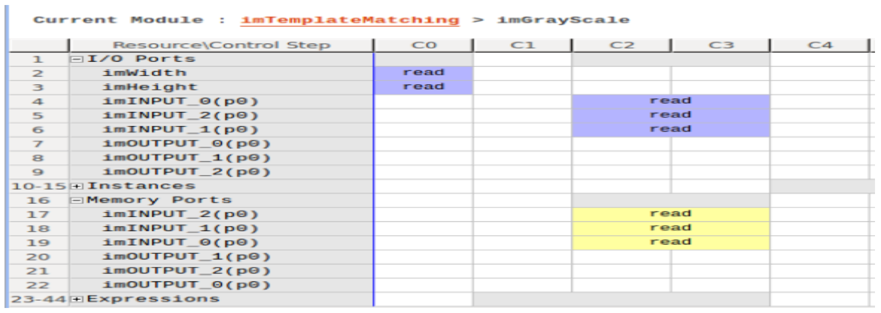
Partitioning the arrays results in an increase of read and write ports for the storage. The RTL design will include multiple small memories and registers instead of one large memory. This method requires more memory instances or registers. Resource utilization increase up to 24.3 % but the performance improves with 51 %. Regarding the performance increase one has to keep in mind that cumulates with the later optimization method which was 14 %. It can be seen from Figure 8 and Figure 9 the impact of array partitioning with a factor of 3.



Current Module : imTemplateMatching > imGrayScale

	Resource\Control Step	C0	C1	C2	C3	C4	C5	C6	C7	C8
1	⊟I/O Ports									
2	imHeight	read								
3	imWidth	read								
4	imINPUT(p0)			read						
5	imOUTPUT(p0)									
6-11	⊞Instances									
12	⊟Memory Ports									
13	imINPUT(p0)			read						
14	imOUTPUT(p0)									
15-30	⊞Expressions									

Figure 8. Array partitioning impact on performance and utilization. Memory port read.

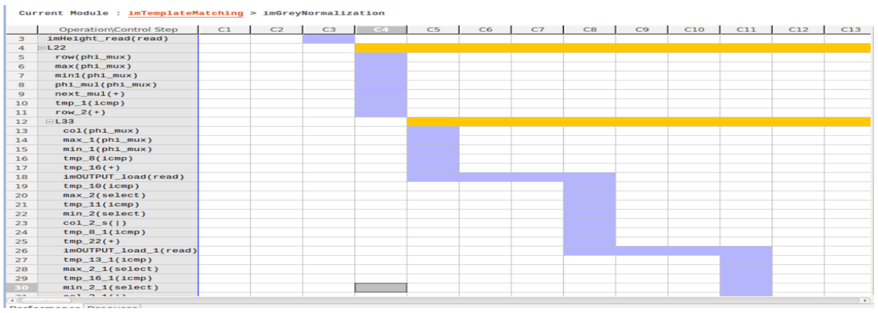


Current Module : imTemplateMatching > imGrayScale

	Resource\Control Step	C0	C1	C2	C3	C4
1	⊟I/O Ports					
2	imWidth	read				
3	imHeight	read				
4	imINPUT_0(p0)			read		
5	imINPUT_2(p0)			read		
6	imINPUT_1(p0)			read		
7	imOUTPUT_0(p0)					
8	imOUTPUT_1(p0)					
9	imOUTPUT_2(p0)					
10-15	⊞Instances					
16	⊟Memory Ports					
17	imINPUT_2(p0)			read		
18	imINPUT_1(p0)			read		
19	imINPUT_0(p0)			read		
20	imOUTPUT_1(p0)					
21	imOUTPUT_2(p0)					
22	imOUTPUT_0(p0)					
23-44	⊞Expressions					

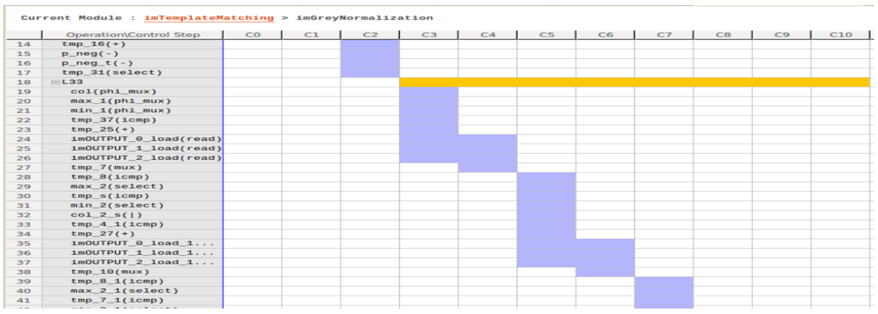
Figure 9. Array partitioning impact on performance and utilization. Memory port read.

Figure 10 shows the performance and control steps of one particular loop from imGreyNormalization function after pragma unroll has benn used. Further optimization can be done if one utilize array partitioning. The differences between Figure 10 and Figure 11 are visible, the number of control steps decrease when array partitioning is used.



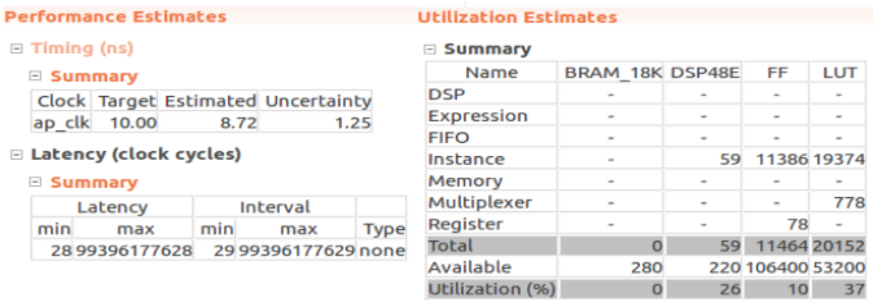
```
Current Module : imTemplateMatching > imGreyNormalization
   Operation\Control Step      C1  C2  C3  C4  C5  C6  C7  C8  C9  C10  C11  C12  C13
3    imHeight_read(read)
4  ⊟L22
5    row(phi_mux)
6    max(phi_mux)
7    min1(phi_mux)
8    phi_mul(phi_mux)
9    next_mul(+)
10   tmp_1(icmp)
11   row_2(+)
12  ⊟L33
13     col(phi_mux)
14     max_1(phi_mux)
15     min_1(phi_mux)
16     tmp_8(icmp)
17     tmp_16(+)
18     imOUTPUT_load(read)
19     tmp_10(icmp)
20     max_2(select)
21     tmp_11(icmp)
22     min_2(select)
23     col_2_s(|)
24     tmp_8_1(icmp)
25     tmp_22(+)
26     imOUTPUT_load_1(read)
27     tmp_13_1(icmp)
28     max_2_1(select)
29     tmp_16_1(icmp)
30     min_2_1(select)
```

Figure 10. Loop unrolling impact on performance.



```
Current Module : imTemplateMatching > imGreyNormalization
   Operation\Control Step      C0  C1  C2  C3  C4  C5  C6  C7  C8  C9  C10
14   tmp_16(+)
15   p_neg(-)
16   p_neg_t(-)
17   tmp_31(select)
18  ⊟L33
19     col(phi_mux)
20     max_1(phi_mux)
21     min_1(phi_mux)
22     tmp_37(icmp)
23     tmp_25(+)
24     imOUTPUT_0_load(read)
25     imOUTPUT_1_load(read)
26     imOUTPUT_2_load(read)
27     tmp_7(mux)
28     tmp_8(icmp)
29     max_2(select)
30     tmp_s(icmp)
31     min_2(select)
32     col_2_s(|)
33     tmp_4_1(icmp)
34     tmp_27(+)
35     imOUTPUT_0_load_1...
36     imOUTPUT_1_load_1...
37     imOUTPUT_2_load_1...
38     tmp_10(mux)
39     tmp_8_1(icmp)
40     max_2_1(select)
41     tmp_7_1(icmp)
```

Figure 11. Loop unrolling and array partitioning impact on performance.



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.72	1.25

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
28	99396177628	29	99396177629	none

Utilization Estimates

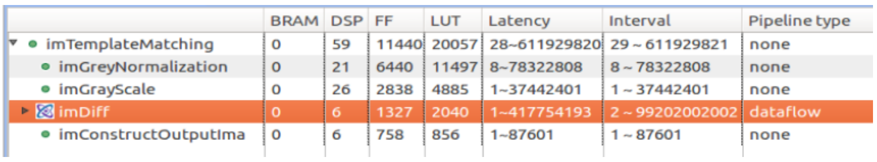
⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	59	11386	19374
Memory	-	-	-	-
Multiplexer	-	-	-	778
Register	-	-	78	-
Total	0	59	11464	20152
Available	280	220	106400	53200
Utilization (%)	0	26	10	37

Figure 12. Array partitioning impact on performance and utilization.

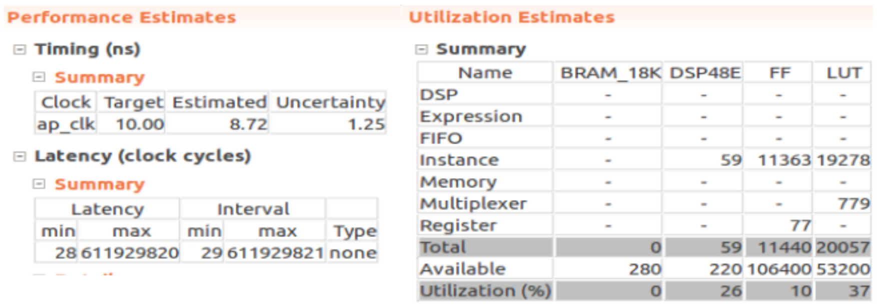
2.2.3. Dataflow

Vivado HLS analyzes the dataflow between sequential functions or loops and create channels (based on pingpong RAMs or FIFOs) that allow consumer functions or loops to start operation before the producer functions or loops have completed. This allows functions or loops to operate in parallel, which decreases latency and improves the throughput of the RTL.



	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
imTemplateMatching	0	59	11440	20057	28~611929820	29 ~ 611929821	none
imGreyNormalization	0	21	6440	11497	8~78322808	8 ~ 78322808	none
imGrayScale	0	26	2838	4885	1~37442401	1 ~ 37442401	none
imDiff	0	6	1327	2040	1~417754193	2 ~ 99202002002	dataflow
imConstructOutputIma	0	6	758	856	1~87601	1 ~ 87601	none

Figure 13. Loop unrolling impact on performance and utilization.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.72	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
28	611929820	29	611929821	none

Utilization Estimates

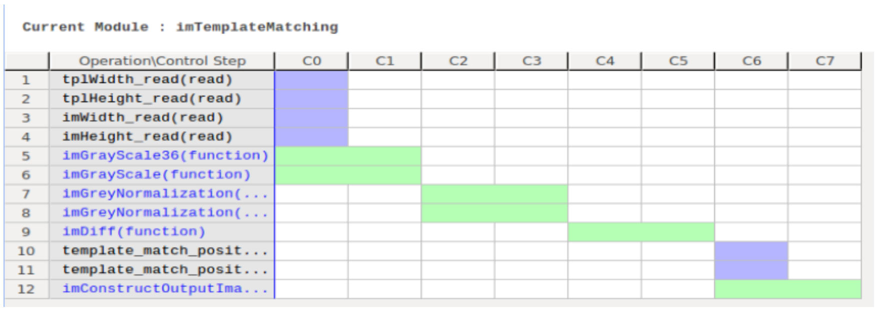
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	59	11363	19278
Memory	-	-	-	-
Multiplexer	-	-	-	779
Register	-	-	77	-
Total	0	59	11440	20057
Available	280	220	106400	53200
Utilization (%)	0	26	10	37

Figure 14. Loop unrolling impact on performance and utilization.

2.2.4. Stream

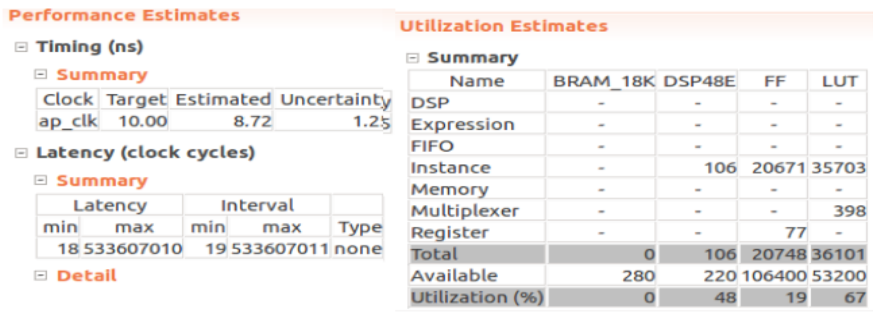
Top-level function array parameters are implemented as a RAM interface port when STREAM directive is used. I have used this directive inside imTemplateMatching function which is the top function. If the data stored in the array is consumed or produced in a sequential manner, a more efficient communication mechanism is to use streaming data as specified by the STREAM pragma, where FIFOs are used instead of RAMs. To see the difference one has to compare Figure 4 with Figure 15. Now the gray scale transformation for both input images (original, template) occurs at the same time in C0 and C1 steps. This applies also for the normalization algorithm. The final performance improves with (cumulated and compared with no optimization solution) is 99 % but the resource utilization is 44.6 %.



Current Module : imTemplateMatching

	Operation\Control Step	C0	C1	C2	C3	C4	C5	C6	C7
1	tplWidth_read(read)	🟦							
2	tplHeight_read(read)	🟦							
3	imWidth_read(read)	🟦							
4	imHeight_read(read)	🟦							
5	imGrayScale36(function)	🟩	🟩						
6	imGrayScale(function)	🟩	🟩						
7	imGreyNormalization(...			🟩	🟩				
8	imGreyNormalization(...			🟩	🟩				
9	imDiff(function)					🟩	🟩		
10	template_match_posit...							🟦	
11	template_match_posit...							🟦	
12	imConstructOutputIma...							🟩	🟩

Figure 15. Streaming data - impact on performance.



Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.72	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
18	533607010	19	533607011	none

Detail

Utilization Estimates

Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	106	20671	35703
Memory	-	-	-	-
Multiplexer	-	-	-	398
Register	-	-	77	-
Total	0	106	20748	36101
Available	280	220	106400	53200
Utilization (%)	0	48	19	67

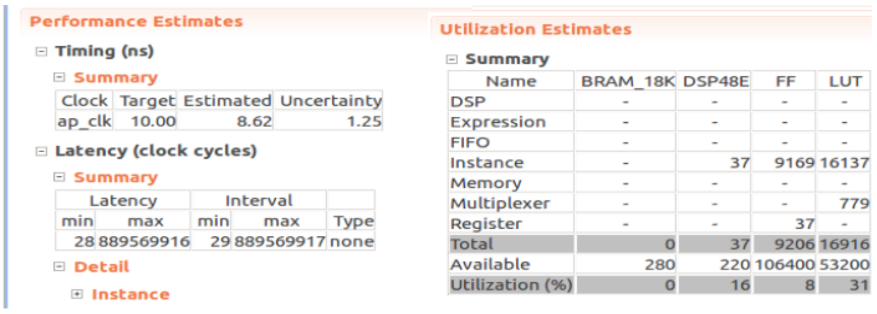
Figure 16. Streaming data - impact on performance and utilization.

2.3. *Area optimization*

As seen before, one has to find the balance between performance and resource utilization. To find the sweet spot between the two some method for area optimization have been used.

2.3.1. Allocation

Allocation limits the number of RTL instances and hardware resources used to implement specific. Because I have function instances that are duplicate, to reduce area I can limit the number of instances to 1 utilizing allocation. This reduces resources utilized by the function, but negatively impacts performance. This method reduces dramatically the resource usage to 18 % and decrease the performance just with 1 % (all the other optimization directives are still applied).


Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.62	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
28	889569916	29	889569917	none

Detail

Instance

Utilization Estimates

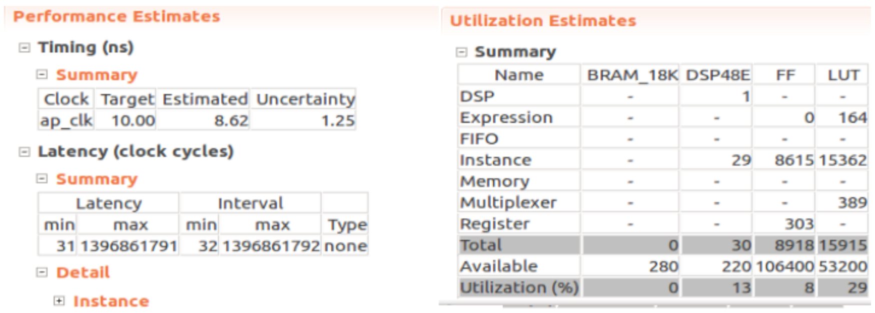
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	-	-	-
Expression	-	-	-	-
FIFO	-	-	-	-
Instance	-	37	9169	16137
Memory	-	-	-	-
Multiplexer	-	-	-	779
Register	-	-	37	-
Total	0	37	9206	16916
Available	280	220	106400	53200
Utilization (%)	0	16	8	31

Figure 17. Allocation - impact on performance and utilization.

2.3.2. Array mapping

Resource consumption can be reduced up to 16 % if array mapping is used. Array mapping directive combines multiple smaller arrays into a single larger array. This larger array can then be targeted to a single larger memory (RAM or FIFO) resource. The performance decreases but not too much compared to last method.


Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.62	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
31	1396861791	32	1396861792	none

Detail

Instance

Utilization Estimates

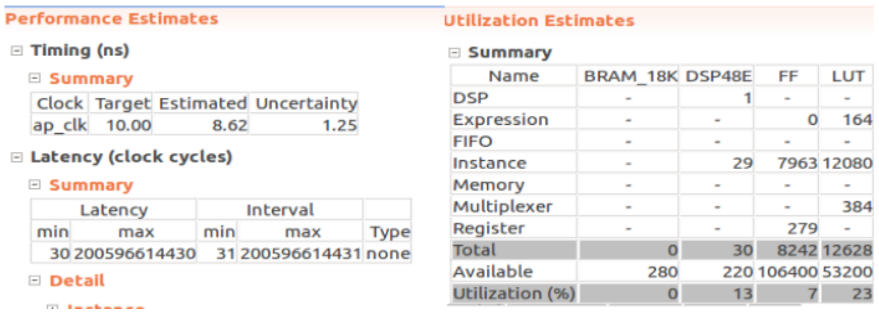
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	164
FIFO	-	-	-	-
Instance	-	29	8615	15362
Memory	-	-	-	-
Multiplexer	-	-	-	389
Register	-	-	303	-
Total	0	30	8918	15915
Available	280	220	106400	53200
Utilization (%)	0	13	8	29

Figure 18. Array mapping - impact on performance and utilization.

2.3.3. Bit-accurate types

Another two area optimization methods have been used but starting from the default solution. In the previous methods the area optimization started from the solution that targets the highest performance and the purpose was to find the sweet spot between performance and area. The default area consumption is around 17 %. Using bit-accurate types for some variables declared inside each function such as indexes the area consumption decrease to 14.5 % and the performance is the same.


Performance Estimates

Timing (ns)

Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.62	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
30	200596614430	31	200596614431	none

Detail

Utilization Estimates

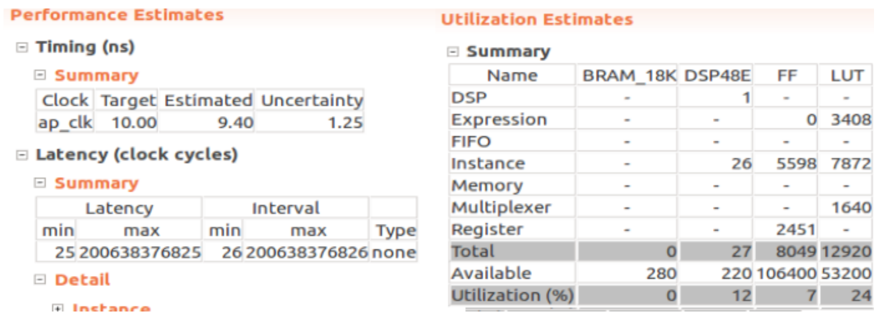
Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	164
FIFO	-	-	-	-
Instance	-	29	7963	12080
Memory	-	-	-	-
Multiplexer	-	-	-	384
Register	-	-	279	-
Total	0	30	8242	12628
Available	280	220	106400	53200
Utilization (%)	0	13	7	23

Figure 19. Bit-accurate types - impact on performance and utilization.

2.3.4. Inline

Other method proposed is to inline the imGrayScale and imNormalization instances. The resource utilization drops to 14.3 %.
Unfortunately no other methods were found to decrease considerably the resource consumption starting from the default optimization. As presented, the decrease in the resource consumption was higher when the performance optimization was used as a starting point.



Performance Estimates

⊟ Timing (ns)

⊟ Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	9.40	1.25

⊟ Latency (clock cycles)

⊟ Summary

Latency		Interval		
min	max	min	max	Type
25	200638376825	26	200638376826	none

⊟ Detail

⊞ Instance

Utilization Estimates

⊟ Summary

Name	BRAM_18K	DSP48E	FF	LUT
DSP	-	1	-	-
Expression	-	-	0	3408
FIFO	-	-	-	-
Instance	-	26	5598	7872
Memory	-	-	-	-
Multiplexer	-	-	-	1640
Register	-	-	2451	-
Total	0	27	8049	12920
Available	280	220	106400	53200
Utilization (%)	0	12	7	24

Figure 20. Inlining - impact on performance and utilization.

References

1. Vivado HLS Optimization Methodology Guide, UG1270 (v2017.4) December 20, 2017
2. Vivado HLS – Tips and Tricks, Xilinx
3. Vivado Design Suite User Guide, UG902 (v2019.1) July 12, 2019
4. Multiprocessors and Reconfigurable Architectures course and lab materials