EE216 Re-configurable Computing Project Report

Zhaojun Ni, 2022231102

January 19, 2024

1 Performance and Resource Utilization

After increasing the frequency to **300MHz**, the latency for processing a single 480p image from TUM RGB-D Dataset on the ZCU104 Evaluation board is only **1.78ms**. It uses 40578 LUTs, 73346 FFs, 149 BRAMs and 193 DSPs.

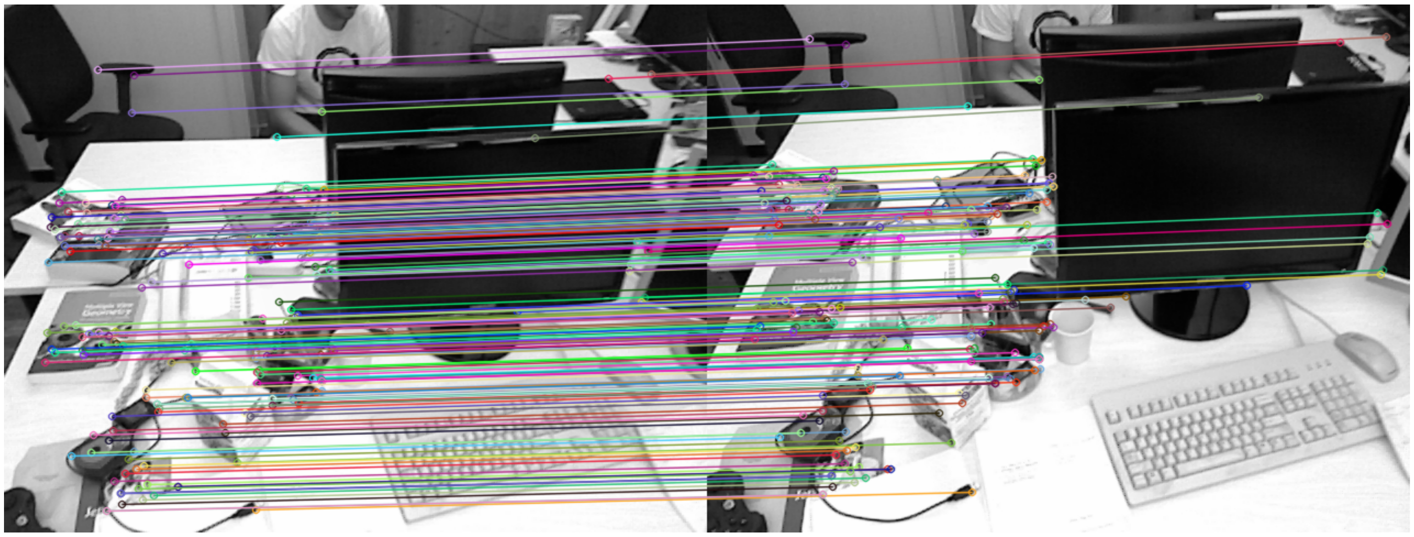
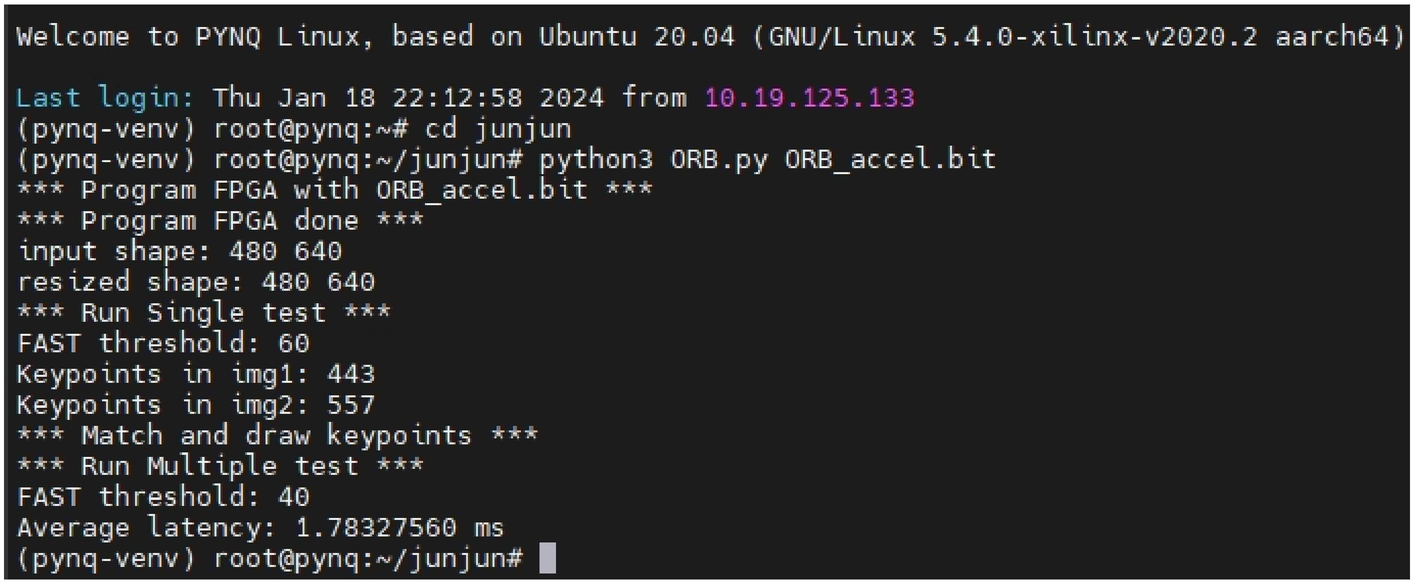


Figure 1: Matching result on ZCU104 Board @ 300MHz.

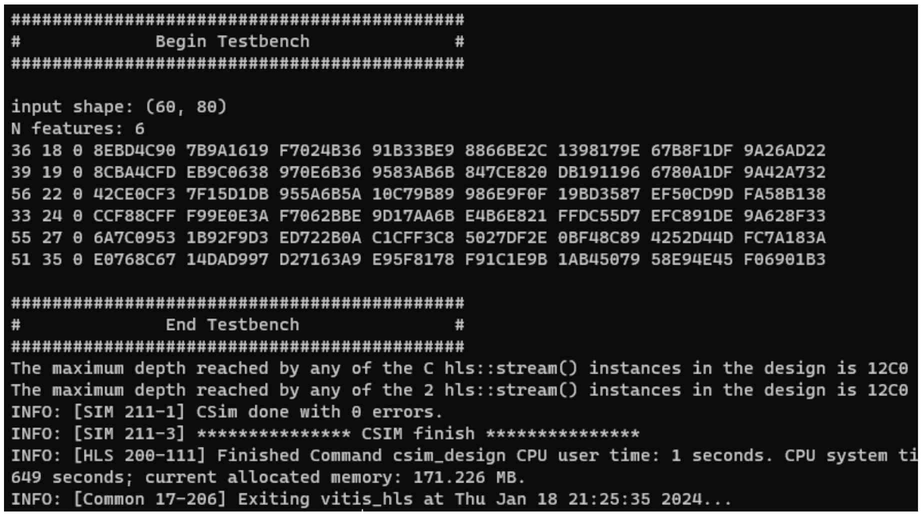


```
Welcome to PYNQ Linux, based on Ubuntu 20.04 (GNU/Linux 5.4.0-xilinx-v2020.2 aarch64)

Last login: Thu Jan 18 22:12:58 2024 from 10.19.125.133
(pynq-venv) root@pynq:~# cd junjun
(pynq-venv) root@pynq:~/junjun# python3 ORB.py ORB_accel.bit
*** Program FPGA with ORB_accel.bit ***
*** Program FPGA done ***
input shape: 480 640
resized shape: 480 640
*** Run Single test ***
FAST threshold: 60
Keypoints in img1: 443
Keypoints in img2: 557
*** Match and draw keypoints ***
*** Run Multiple test ***
FAST threshold: 40
Average latency: 1.78327560 ms
(pynq-venv) root@pynq:~/junjun#
```

Figure 2: Pynq running result on ZCU104 Board @ 300MHz.

2 Experimental Records

# 1. C Simulation

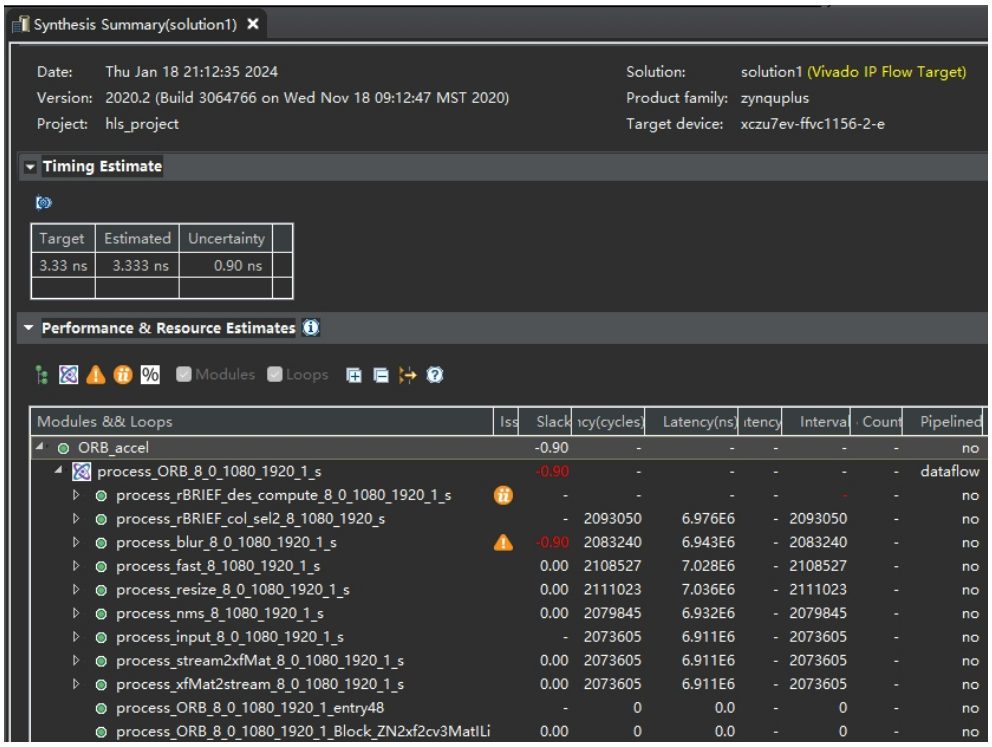


```
############################################
#              Begin Testbench             #
############################################

input shape: (60, 80)
N features: 6
36 18 0 8EBD4C90 7B9A1619 F7024B36 91B33BE9 8866BE2C 1398179E 67B8F1DF 9A26AD22
39 19 0 8CBA4CFD EB9C0638 970E6B36 9583AB6B 847CE820 DB191196 6780A1DF 9A42A732
56 22 0 42CE0CF3 7F15D1DB 955A6B5A 10C79B89 986E9F0F 19BD3587 EF50CD9D FA58B138
33 24 0 CCF88CFF F99E0E3A F7062BBE 9D17AA6B E4B6E821 FFDC55D7 EFC891DE 9A628F33
55 27 0 6A7C0953 1B92F9D3 ED722B0A C1CFF3C8 5027DF2E 0BF48C89 4252D44D FC7A183A
51 35 0 E0768C67 14DAD997 D27163A9 E95F8178 F91C1E9B 1AB45079 58E94E45 F06901B3

############################################
#               End Testbench              #
############################################
The maximum depth reached by any of the C hls::stream() instances in the design is 12C0
The maximum depth reached by any of the 2 hls::stream() instances in the design is 12C0
INFO: [SIM 211-1] CSim done with 0 errors.
INFO: [SIM 211-3] *************** CSIM finish ***************
INFO: [HLS 200-111] Finished Command csim_design CPU user time: 1 seconds. CPU system ti
649 seconds; current allocated memory: 171.226 MB.
INFO: [Common 17-206] Exiting vitis_hls at Thu Jan 18 21:25:35 2024...
```

Figure 3: Csim Result

# 2. Synthesis

The synthesis report shows negative slack which equals to the uncertainty while it works fine on the board. And the interval column shows that the system has balanced throughput. And it indicates that 300 MHz may be the peak frequency of the design.



Synthesis Summary(solution1)

Date:	Thu Jan 18 21:12:35 2024	Solution:	solution1 (Vivado IP Flow Target)
Version:	2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)	Product family:	zynquplus
Project:	hls_project	Target device:	xczu7ev-ffvc1156-2-e

Timing Estimate

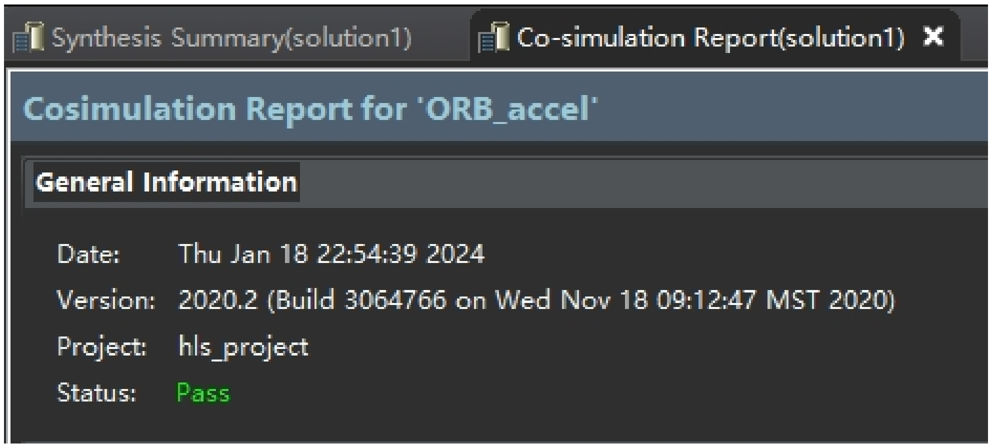
Target	Estimated	Uncertainty
3.33 ns	3.333 ns	0.90 ns

Performance & Resource Estimates

Modules && Loops	Iss	Slack	ıcy(cycles)	Latency(ns)	ıtency	Interval	Count	Pipelined
ORB_accel		-0.90	-	-	-	-	-	no
process_ORB_8_0_1080_1920_1_s		-0.90	-	-	-	-	-	dataflow
process_rBRIEF_des_compute_8_0_1080_1920_1_s		-	-	-	-	-	-	no
process_rBRIEF_col_sel2_8_1080_1920_s		-	2093050	6.976E6	-	2093050	-	no
process_blur_8_0_1080_1920_1_s		-0.90	2083240	6.943E6	-	2083240	-	no
process_fast_8_1080_1920_1_s		0.00	2108527	7.028E6	-	2108527	-	no
process_resize_8_0_1080_1920_1_s		0.00	2111023	7.036E6	-	2111023	-	no
process_nms_8_1080_1920_1_s		0.00	2079845	6.932E6	-	2079845	-	no
process_input_8_0_1080_1920_1_s		-	2073605	6.911E6	-	2073605	-	no
process_stream2xfMat_8_0_1080_1920_1_s		0.00	2073605	6.911E6	-	2073605	-	no
process_xfMat2stream_8_0_1080_1920_1_s		0.00	2073605	6.911E6	-	2073605	-	no
process_ORB_8_0_1080_1920_1_entry48		-	0	0.0	-	0	-	no
process_ORB_8_0_1080_1920_1_Block_ZN2xf2cv3MatILi		0.00	0	0.0	-	0	-	no

Figure 4: Synthesis Report

# 3. Co-Simulation


Synthesis Summary(solution1)   Co-simulation Report(solution1)

Cosimulation Report for 'ORB_accel'

General Information

Date: Thu Jan 18 22:54:39 2024
Version: 2020.2 (Build 3064766 on Wed Nov 18 09:12:47 MST 2020)
Project: hls_project
Status: Pass

Figure 5: Co-sim Report

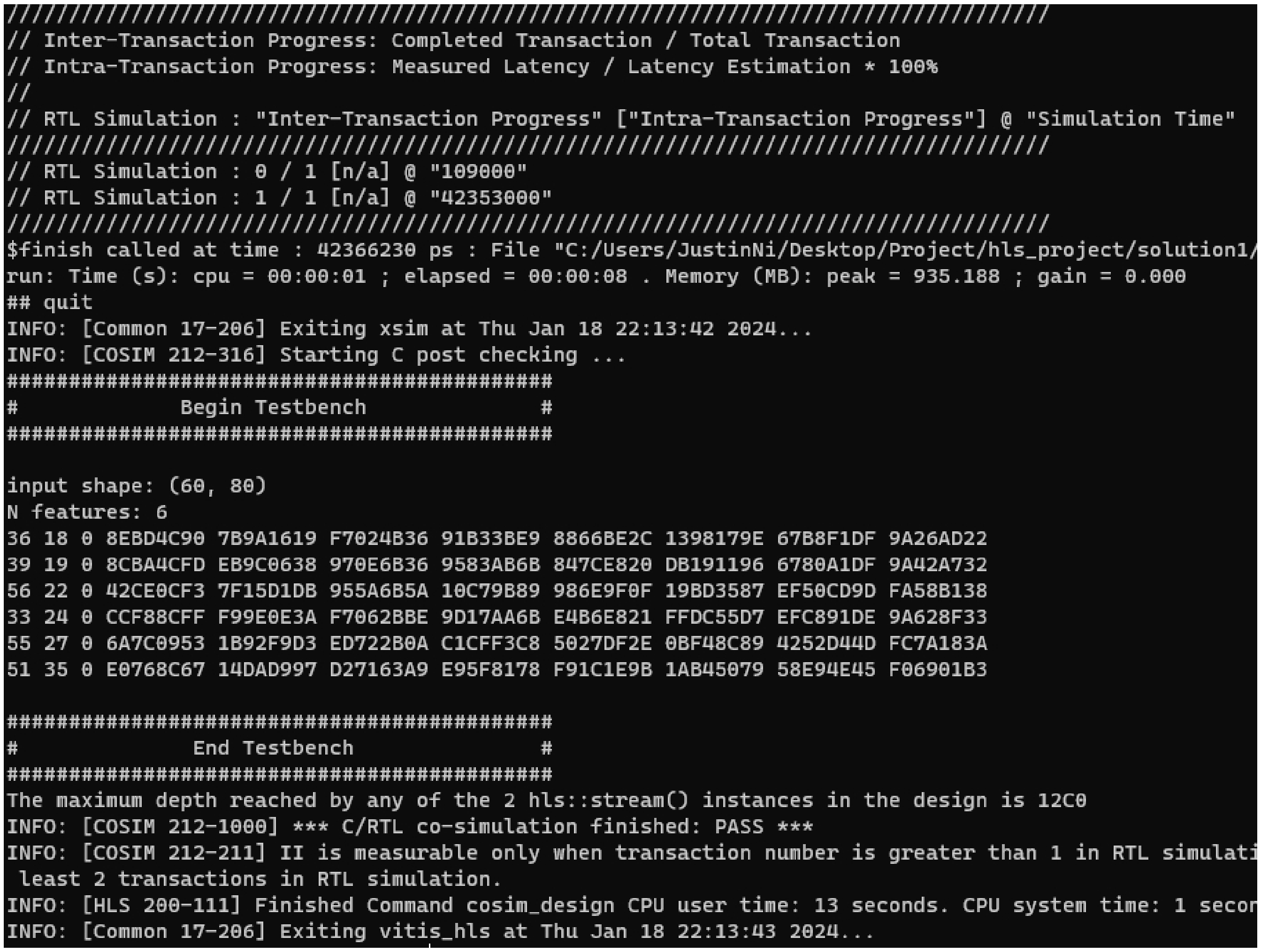

```
////////////////////////////////////////////////////////////////////////////////
// Inter-Transaction Progress: Completed Transaction / Total Transaction
// Intra-Transaction Progress: Measured Latency / Latency Estimation * 100%
//
// RTL Simulation : "Inter-Transaction Progress" ["Intra-Transaction Progress"] @ "Simulation Time"
////////////////////////////////////////////////////////////////////////////////
// RTL Simulation : 0 / 1 [n/a] @ "109000"
// RTL Simulation : 1 / 1 [n/a] @ "42353000"
////////////////////////////////////////////////////////////////////////////////
$finish called at time : 42366230 ps : File "C:/Users/JustinNi/Desktop/Project/hls_project/solution1/
run: Time (s): cpu = 00:00:01 ; elapsed = 00:00:08 . Memory (MB): peak = 935.188 ; gain = 0.000
## quit
INFO: [Common 17-206] Exiting xsim at Thu Jan 18 22:13:42 2024...
INFO: [COSIM 212-316] Starting C post checking ...
###########################################
#             Begin Testbench             #
###########################################

input shape: (60, 80)
N features: 6
36 18 0 8EBD4C90 7B9A1619 F7024B36 91B33BE9 8866BE2C 1398179E 67B8F1DF 9A26AD22
39 19 0 8CBA4CFD EB9C0638 970E6B36 9583AB6B 847CE820 DB191196 6780A1DF 9A42A732
56 22 0 42CE0CF3 7F15D1DB 955A6B5A 10C79B89 986E9F0F 19BD3587 EF50CD9D FA58B138
33 24 0 CCF88CFF F99E0E3A F7062BBE 9D17AA6B E4B6E821 FFDC55D7 EFC891DE 9A628F33
55 27 0 6A7C0953 1B92F9D3 ED722B0A C1CFF3C8 5027DF2E 0BF48C89 4252D44D FC7A183A
51 35 0 E0768C67 14DAD997 D27163A9 E95F8178 F91C1E9B 1AB45079 58E94E45 F06901B3

###########################################
#              End Testbench              #
###########################################
The maximum depth reached by any of the 2 hls::stream() instances in the design is 12C0
INFO: [COSIM 212-1000] *** C/RTL co-simulation finished: PASS ***
INFO: [COSIM 212-211] II is measurable only when transaction number is greater than 1 in RTL simulati
 least 2 transactions in RTL simulation.
INFO: [HLS 200-111] Finished Command cosim_design CPU user time: 13 seconds. CPU system time: 1 secor
INFO: [Common 17-206] Exiting vitis_hls at Thu Jan 18 22:13:43 2024...
```

Figure 6: Co-sim Result

3 Optimization Methods

# 0. Loop Tripcount

In HLS, specifying the tripcount for a loop is the first step in optimization. This allows the synthesis report to show the expected number of cycles each function is likely to consume. Based on the order of magnitude of the cycle count in the synthesis report, along with the relationship between frequency and the proportion of processed images, one can roughly estimate the delay.

# 1. Loop Unroll

Loop unrolling in HLS is an optimization technique that involves expanding the loop by replicating the loop body's iterations, aiming to reduce loop control overhead and enhance instruction-level parallelism. In certain situations, this can improve the performance of hardware descriptions. For example, in "fast.hpp", loop unrolling has been applied to the loop inside the "fast_score" function. Similarly, in the "process_fast" function, loop unrolling has been performed on the explicitly bounded "for" loop, with the additional specification of the unroll factor. The "unroll factor" is the number of times a loop is unfolded, indicating the repetition count of each original iteration in the unfolded copies.

# 2. Array Partition

Array partitioning is used to divide an array, allowing the partitioned sections to be read within a single clock cycle. For instance, when dealing with a line buffer that performs row-wise read operations, and considering a two-dimensional array where the first dimension represents rows and the second dimension represents columns, performing a complete partition on the first dimension is sufficient. Similarly, for small windows like a 7x7 convolution kernel, complete partitioning can be done in both dimensions.

In the "loop_descriptor", I observed potential conflicts between accesses by "t0" and "t1" to the "window". To address this, I created two copies, "window_buf0" and "buf1", dedicated to "t0" and "t1" respectively. When attempting complete partitioning in both dimensions for these two buffers, the synthesis process became very slow. To mitigate this, I explicitly specified partitioning in only the second dimension for both buffers. Otherwise, synthesis would generate certain warnings.

# 3. Pipeline

In this project, multiple functions starting with "process" are annotated with the "pragma HLS DATAFLOW". Consequently, from the synthesis report, it is observed that modules traversed by DATAFLOW seem to undergo automatic pipelining within their internals after adjusting loop boundaries. For certain pixel traversal operations within functions, such as iterating over columns, the report indicates that pipelining is enabled (pipeline: yes).

# 4. Non-Blocking Optimization

From the initial synthesis report, it is evident that the main delay occurs in the "process_rBRIEF" operation, and it consumes the majority of hardware resources. From the perspective of the "process_rBRIEF" function, it begins by traversing all pixels through two nested loops. If the central pixel of the mask window is identified as a key point, computations for the functions "IC Angle 31" and "descriptor' are then performed. The former, "IC Angle 31", involves several loops with undefined loop boundaries, while the latter, "descriptor", contains an additional loop with 256 iterations.

a. Unroll loops in IC Angle 31.

In this function, the outer loop's iteration count, denoted as variable "v," is stored in an array. I manually unrolled the loop by extracting the values from the 15-column array and individually set them as the loop's upper limit. This way, each sub-loop can be separately specified with an UNROLL FACTOR for parallel computation.

b. Cache window columns[1].

Upon closer inspection, the initial version's slow speed can be attributed to its blocking nature. It requires the completion of a series of operations in "IC Angle 31" and "descriptor" before accessing the next pixel. Drawing inspiration from the referenced literature[1], to achieve non-blocking computation for rBRIEF, a FIFO is employed to cache the window columns. In specific operations, it is possible to modify the original "process_rBRIEF" and "process_output" functions by specifying the output of the former and the input of the latter as HLS stream types.

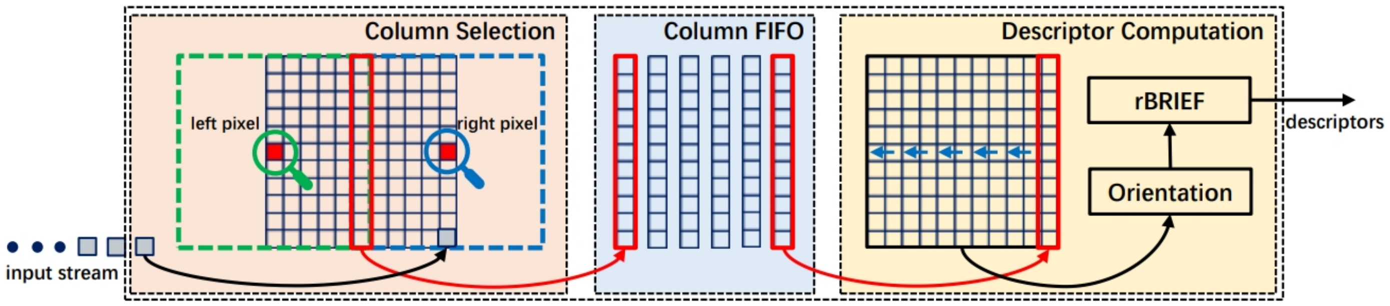


Figure 7: Non-blocking computation is achieved by caching the window columns.[1]

Figure 7 is an rBRIEF module proposed by the authors in Reference 1. I have implemented it using HLS, with the Column Selection part realized in the "process_rBRIEF_col_sel" function and the Descriptor Computation implemented in the "process_rBRIEF_des_compute" function. The data type of the column FIFO in the middle of the figure directly utilizes the "WindowCol" structure from the ORB.h file, and data flow is facilitated through the use of "hls::stream".

The author's idea is that, as a window of length 37 traverses from left to right, attention should be paid to whether there are key points in the leftmost and rightmost columns of the window. If a key point is detected in the rightmost column, the author starts outputting the middle column's window column through a FIFO to the subsequent descriptor computation module. After precisely 36 cycles, 37 columns will be written into the FIFO. At this point, the complete orientation and rBRIEF for the window can be calculated.

The author does not use a counter to determine whether the window is intact. This is because it is possible to encounter two key points in the same row with a separation of less than a window size. The author's idea of detecting the leftmost column of the window is **clever**. This is because the first key point encountered in the rightmost column will naturally move to the position detected by the leftmost column after exactly 36 cycles. At this moment, a column-end label signal can be written into the FIFO to initiate the computation of the key point's window.

This algorithm is advantageous as it utilizes a FIFO. In the window buffer of the descriptor computation, each column shifts to the left every other cycle, and the newest column is the one most recently written. In cases of adjacent key points, this allows for **efficient data reuse**, thereby reducing latency to a significant extent.

c. Implement Column Selection.

My "process_rBRIEF_col_sel" function is essentially implemented according to the algorithm in the author's paper. However, during simulation, I encountered some issues. If a 37x37 window directly slides from the top-left corner of the image to the right, the simulated output only has 2 feature points instead of 6. The reason is that if there is no horizontal padding, the leftmost column written into when the sliding window reaches the position of the 19th column (HALF WINDOW SIZE) – meaning the feature points at the leftmost and rightmost positions of the window – will not be detected.

To address this, I introduced padding in the column traversal of the algorithm. I filled the leftmost 18 columns and the rightmost 18 columns with zeros. After making this modification, the simulation results were error-free, and it correctly outputted all six feature points.

d. Column FIFO data component.

The first three bits of this structure serve as flags, representing the "last," "begin," and "end" signals. The "last" flag indicates whether the current column being written is the last one in the entire image. The "begin" flag indicates whether the current column is the first column in the key point window. The "end" flag indicates whether the current column is the last column in the key point window. Following the flags, there are two uint16 data fields representing the coordinates of the key point. Then, there are two uint data fields with a bit width of 8 * WIND_SIZE, representing the columns of the image and blured image. The final uint8 data is the value from the mask window, i.e., the response value of the key point. The depth of FIFO is set to 200 according to the reference[1].

e. Implement Descriptor Computation.

I replaced the original "output" function with my self-implemented "process_rBRIEF_des_compute" function to handle the output of descriptors. Similarly, I use a while loop to continuously process data from the FIFO. As long as the received data's "flag last" is not 1, the processing continues. In each cycle, if the data in the FIFO is valid, the function reads the structure data from the FIFO. Then, it performs a circular left shift on the "img window" and "blur window". Next, it inserts the "img col" and "blur col" from the structure into the last columns of their respective windows. When the read "flag_end" is 1, it indicates that the data in the current window is a complete window for a key point. Subsequently, the angle and descriptor calculations are carried out. The coordinate information, response value, descriptor calculation results, and other details are then written to the "dst_axi". Following the style of the original "process_output" function, the AXI signals "last" are set to 0, "keep" to -1. Additionally, for sorting convenience, the results are written into "idxTree" and "cntTree".

4 Reference

[1]Q. Zhang, H. Sun, Q. Deng, H. Yu and Y. Ha, "NORB: A Stream-Based and Non-Blocking FPGA Accelerator for ORB Feature Extraction," 2023 30th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Istanbul, Turkiye, 2023, pp. 1-4, doi: 10.1109/ICECS58634.2023.10382726.