MSoC Final Presentation
Team19 https://github.com/ycjustin-msoc/team19_final

YOLO：You Only Look Once

You-Huei Chen, Lian-Fu Shen
Department of Engineering Science and Ocean Engineering
National Taiwan University
Bo-An Chen
Graduate Institute of Electronic Engineering
National Taiwan University

Outline

Outline

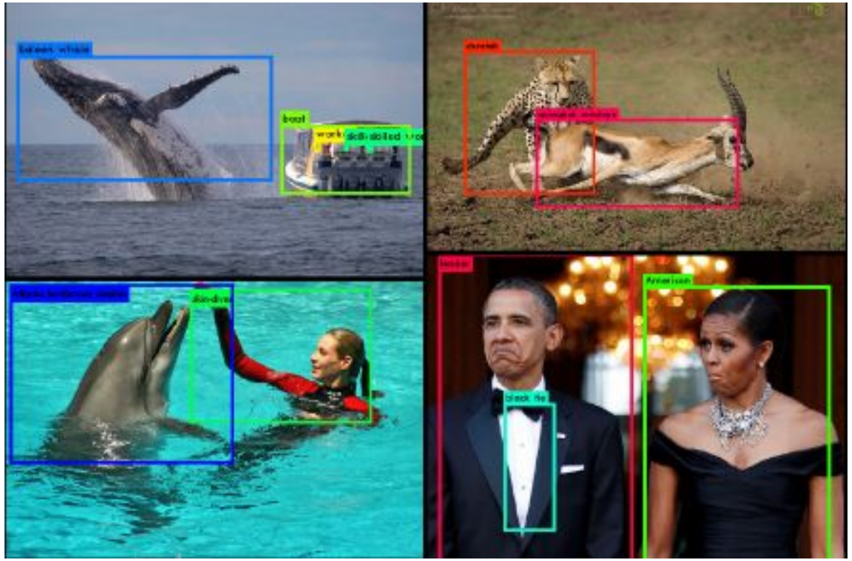
YOLOv2

A.K.A YOLO9000

Joseph Redmon, Ali Farhadi et al., 2016

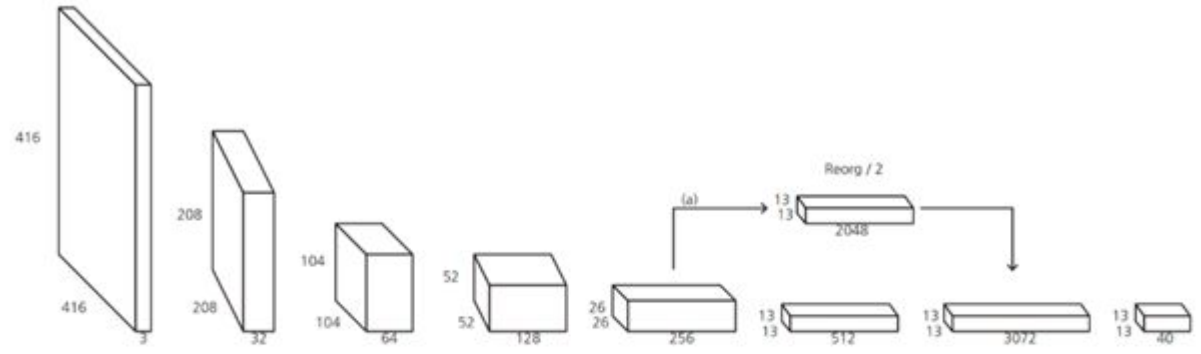
- Better – Highest mAP
- Faster – Highest FPS
- Stronger – More than 9000 classes

	mAP	FPS
Faster R-CNN ResNet [3]	76.4	5
YOLOv1 (488*488) [4]	63.4	45
YOLOv2 (416*416)	76.8	67
YOLOv2 (544*544)	78.6	40

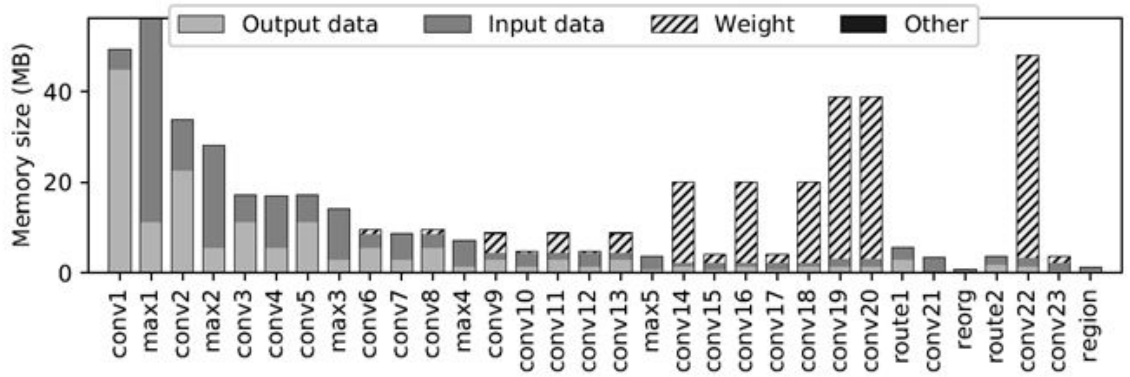


[2]

Computation Complexity



YOLO v2 architecture [5]



Memory footprint [6]

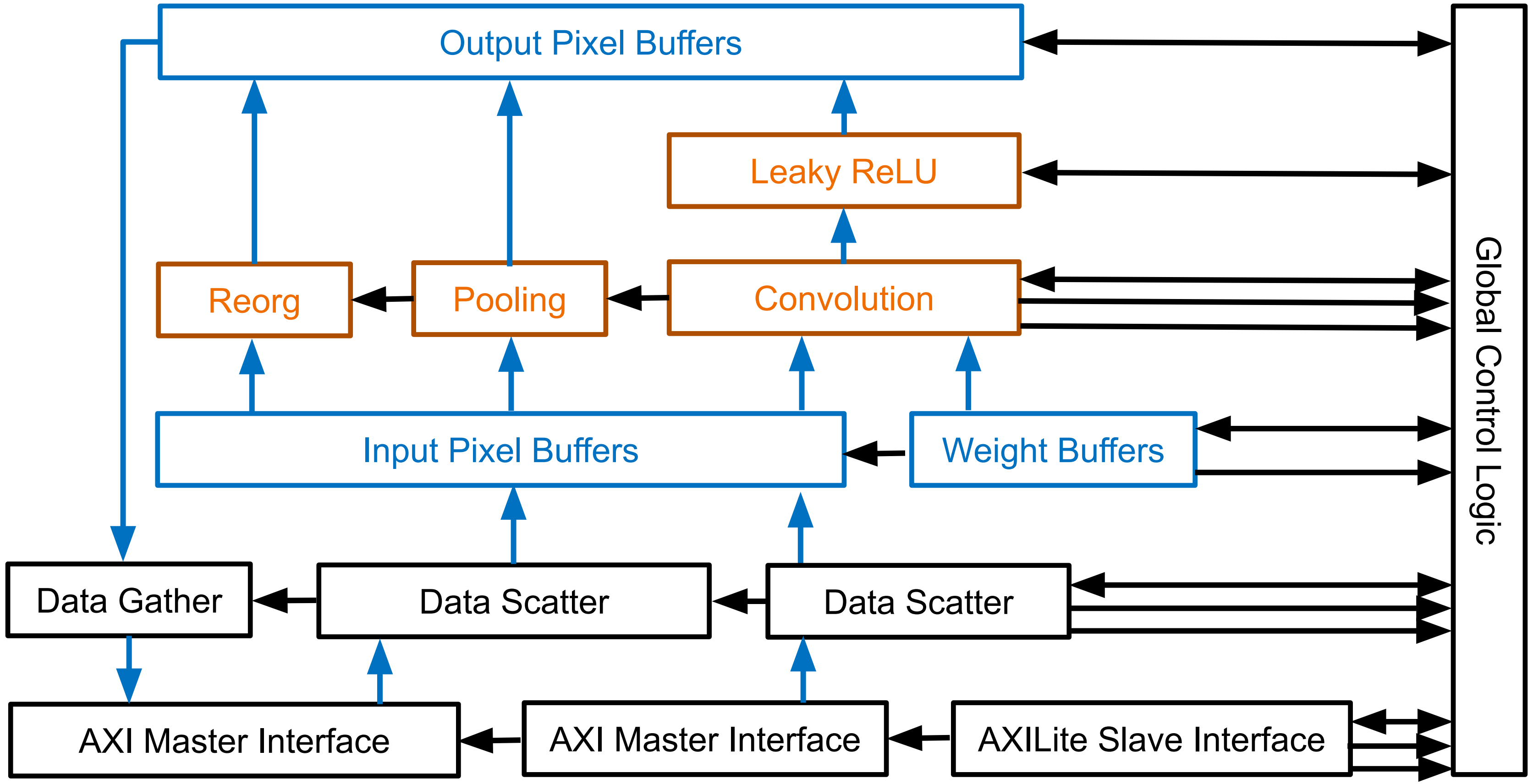
HLS Core Design

- ***Github source code:***
  https://github.com/dhm2013724/yolov2_xilinx_fpga [1]

- ***Target :***
  - ✓ Enable real-time object detections
  - ✓ Meet circuit's timing
  - ✓ Verify this design by FPGA
  - ✓ Achieve 1 FPS

Outline

Block Diagram



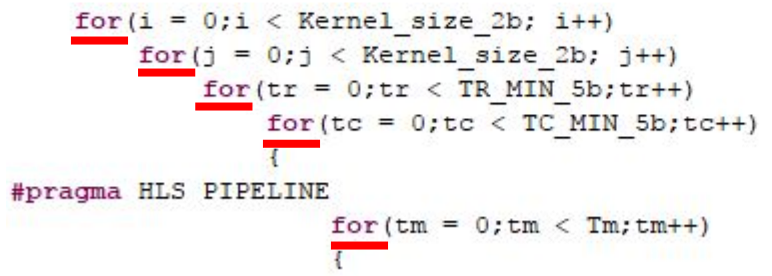
Preparations for Validation

Software Generation

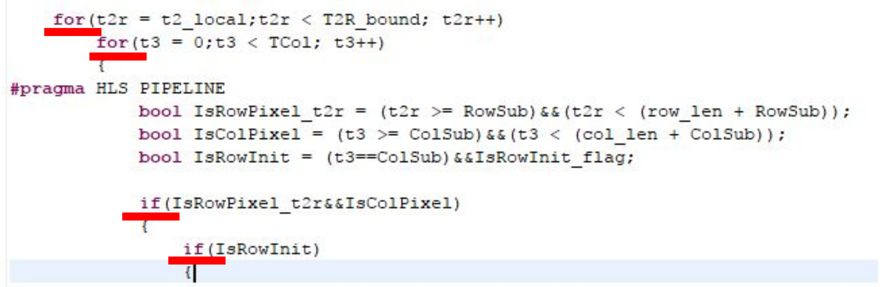
- Extract weight/bias from the Darknet
- Reorganize YOLOv2's weights and quantize weights and biases from float32 to fixed16
- Quantize weights and biases for 23 convolutional layers
- The following weight/bias files are generated and will be applied for verification
    - bias_ap16.bin
    - weights_reorg_ap16.bin
- Extract all variables of YOLOv2 for tripcount estimation

Bottleneck

- Design phase
  - Co-sim can’t be run
  - Csim takes about 50 mins and can’t present HLS pragma result
  - Checking the result of optimization through Vivado and Pynq is the only way!
- Architecture phase
  - Too much for-loops (e.g. a three dimension input_buffer in a 3-stage for-loop)
  - Hard to solve multiple carried dependencies (e.g. nested “If” inside a deep for-loop)



```
    for(i = 0;i < Kernel_size_2b; i++)
        for(j = 0;j < Kernel_size_2b; j++)
            for(tr = 0;tr < TR_MIN_5b;tr++)
                for(tc = 0;tc < TC_MIN_5b;tc++)
                {
#pragma HLS PIPELINE
                    for(tm = 0;tm < Tm;tm++)
                    {
```

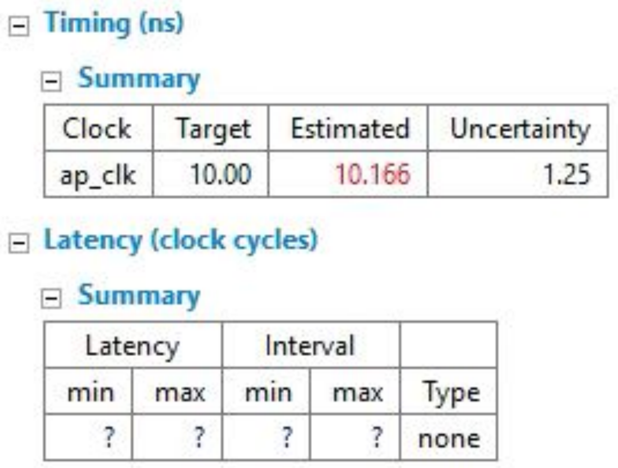


```
  for(t2r = t2_local;t2r < T2R_bound; t2r++)
     for(t3 = 0;t3 < TCol; t3++)
        {
#pragma HLS PIPELINE
            bool IsRowPixel_t2r = (t2r >= RowSub)&&(t2r < (row_len + RowSub));
            bool IsColPixel = (t3 >= ColSub)&&(t3 < (col_len + ColSub));
            bool IsRowInit = (t3==ColSub)&&IsRowInit_flag;

            if(IsRowPixel_t2r&&IsColPixel)
            {
                if(IsRowInit)
                {
```

Baseline Report

- For the baseline report, the design requires 10.166ns
- fpga_process_time: 2.69s@Pynq
- We aim to resolve timing violations and decrease latency



Timing (ns)

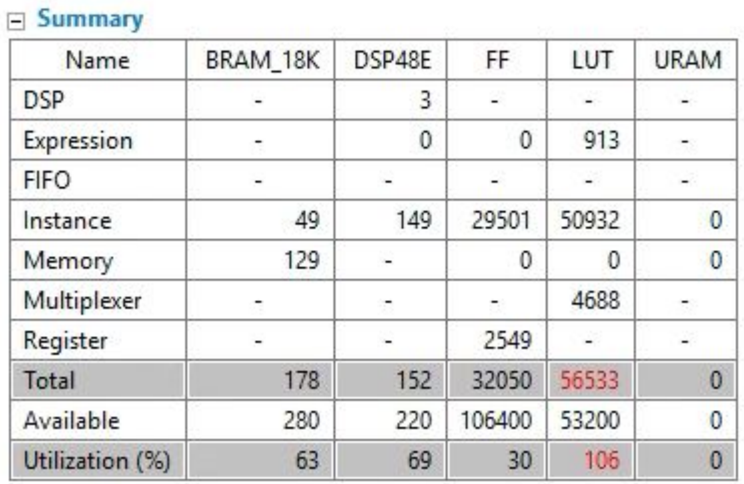
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.166	1.25

Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
?	?	?	?	none



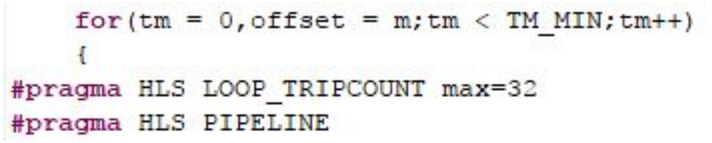
Summary

Name	BRAM_18K	DSP48E	FF	LUT	URAM
DSP	-	3	-	-	-
Expression	-	0	0	913	-
FIFO	-	-	-	-	-
Instance	49	149	29501	50932	0
Memory	129	-	0	0	0
Multiplexer	-	-	-	4688	-
Register	-	-	2549	-	-
Total	178	152	32050	56533	0
Available	280	220	106400	53200	0
Utilization (%)	63	69	30	106	0

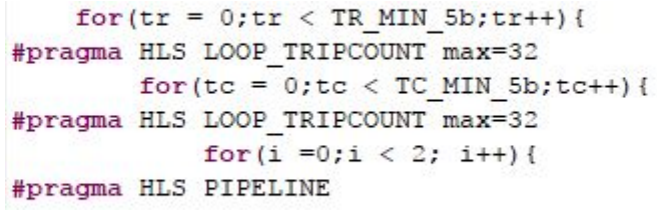
Optimization

Phase 1

- Most of loop's bounding in this design are variables so that the loops could not be pipelined or unrolled. Therefore, we added directives to specify its max latency.
  #Pragma HLS LOOP_TRIPCOUNT max = <max_val>



```
    for(tm = 0,offset = m;tm < TM_MIN;tm++)
    {
#pragma HLS LOOP_TRIPCOUNT max=32
#pragma HLS PIPELINE
```

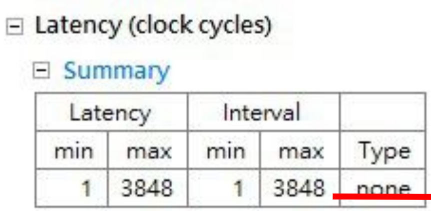


```
    for(tr = 0;tr < TR_MIN_5b;tr++){
#pragma HLS LOOP_TRIPCOUNT max=32
        for(tc = 0;tc < TC_MIN_5b;tc++){
#pragma HLS LOOP_TRIPCOUNT max=32
            for(i =0;i < 2; i++){
#pragma HLS PIPELINE
```

Optimization

Phase 1

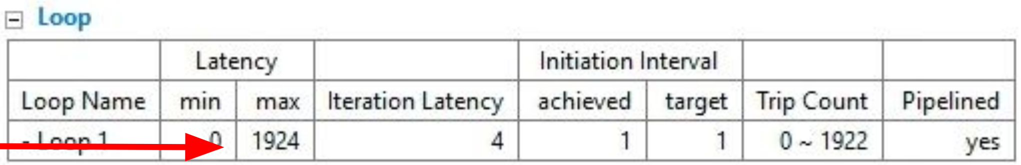
- Achieve “Reorg” and “Pool” functions for pipelined II=1 and decrease the latency
  - “Reorg”



Latency (clock cycles)

Summary

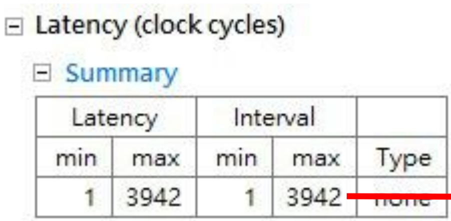
Latency		Interval		
min	max	min	max	Type
1	3848	1	3848	none



Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	0	1924	4	1	1	0 ~ 1922	yes

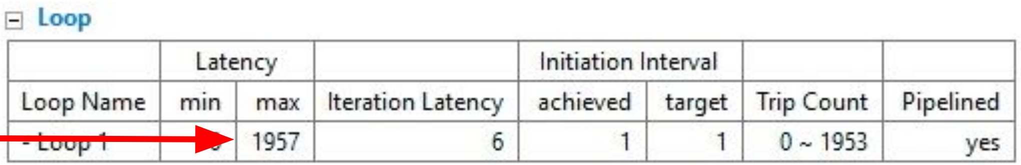
  - “Pool”



Latency (clock cycles)

Summary

Latency		Interval		
min	max	min	max	Type
1	3942	1	3942	none



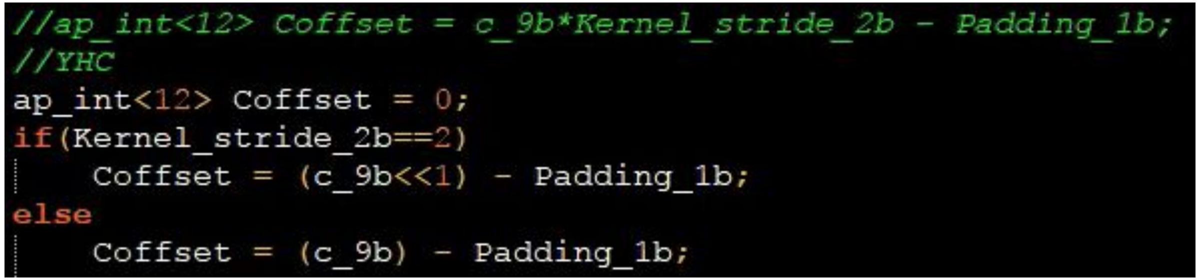
Loop

	Latency			Initiation Interval			
Loop Name	min	max	Iteration Latency	achieved	target	Trip Count	Pipelined
- Loop 1	0	1957	6	1	1	0 ~ 1953	yes

Optimization

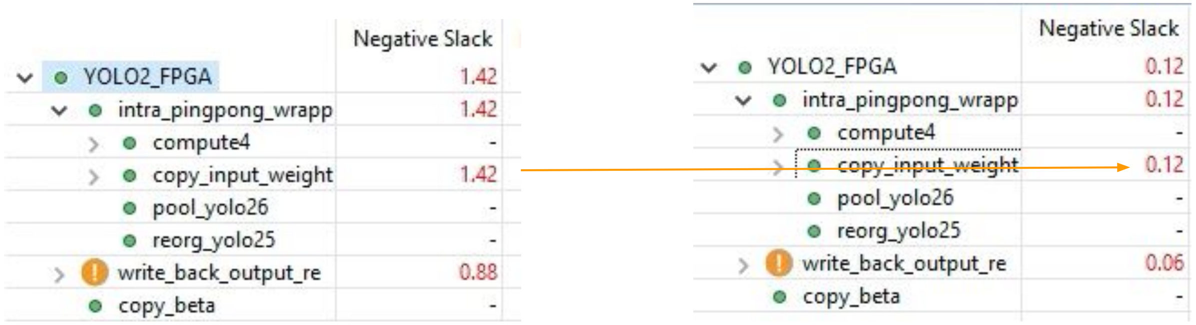
Phase 2

- Solve timing violations by changing coding style



```
//ap_int<12> Coffset = c_9b*Kernel_stride_2b - Padding_1b;
//YHC
ap_int<12> Coffset = 0;
if(Kernel_stride_2b==2)
    Coffset = (c_9b<<1) - Padding_1b;
else
    Coffset = (c_9b) - Padding_1b;
```

- By extracting the variables, we realize that “Kernel_stribe” can only be 1 or 2



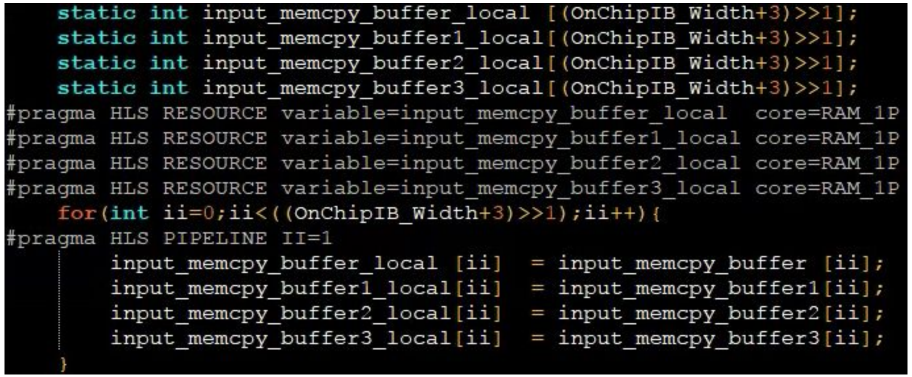
	Negative Slack
YOLO2_FPGA	1.42
intra_pingpong_wrapp	1.42
compute4	-
copy_input_weight	1.42
pool_yolo26	-
reorg_yolo25	-
write_back_output_re	0.88
copy_beta	-

	Negative Slack
YOLO2_FPGA	0.12
intra_pingpong_wrapp	0.12
compute4	-
copy_input_weight	0.12
pool_yolo26	-
reorg_yolo25	-
write_back_output_re	0.06
copy_beta	-

Optimization

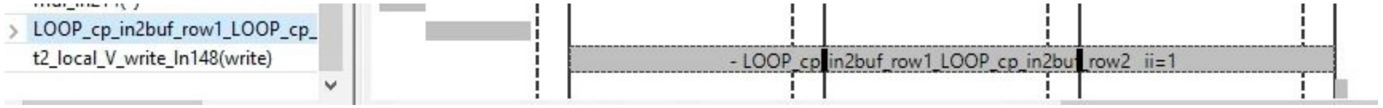
Phase 3

- Using local buffer to store data first can completely transfer one row in a time



```
    static int input_memcpy_buffer_local [(OnChipIB_Width+3)>>1];
    static int input_memcpy_buffer1_local[(OnChipIB_Width+3)>>1];
    static int input_memcpy_buffer2_local[(OnChipIB_Width+3)>>1];
    static int input_memcpy_buffer3_local[(OnChipIB_Width+3)>>1];
#pragma HLS RESOURCE variable=input_memcpy_buffer_local  core=RAM_1P
#pragma HLS RESOURCE variable=input_memcpy_buffer1_local core=RAM_1P
#pragma HLS RESOURCE variable=input_memcpy_buffer2_local core=RAM_1P
#pragma HLS RESOURCE variable=input_memcpy_buffer3_local core=RAM_1P
    for(int ii=0;ii<((OnChipIB_Width+3)>>1);ii++){
#pragma HLS PIPELINE II=1
        input_memcpy_buffer_local [ii]  = input_memcpy_buffer [ii];
        input_memcpy_buffer1_local[ii]  = input_memcpy_buffer1[ii];
        input_memcpy_buffer2_local[ii]  = input_memcpy_buffer2[ii];
        input_memcpy_buffer3_local[ii]  = input_memcpy_buffer3[ii];
    }
```

- After the data transfer between functions decreasing, then we help the following operations achieve “II=1” inside a for-loop

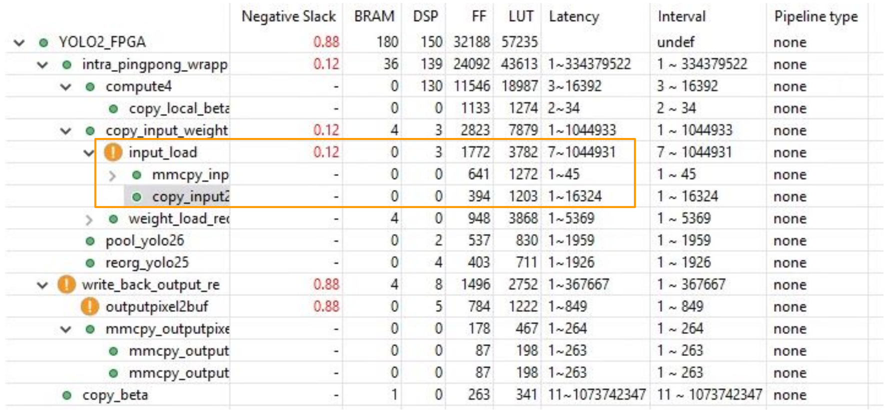


```
> LOOP_cp_in2buf_row1_LOOP_cp_
  t2_local_V_write_ln148(write)
- LOOP_cp_in2buf_row1_LOOP_cp_in2buf_row2  ii=1
```

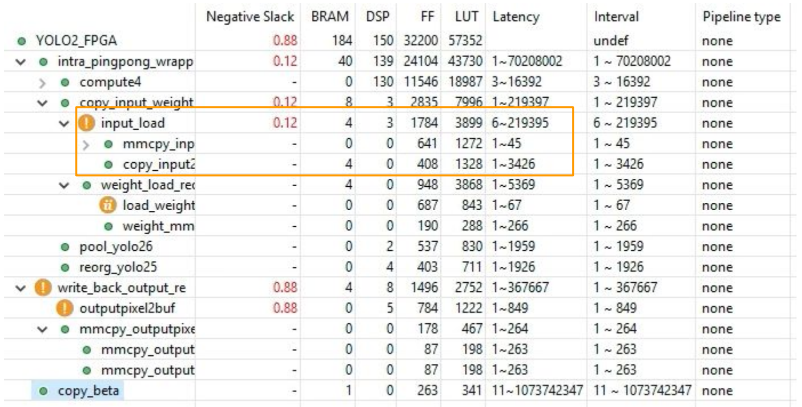
Optimization

Phase 3

- "Input_load" function is the biggest part of "Latency", and each layer of operations will execute "Input_load" one time.
- Decrease "Input_load" latency by "4.7X".



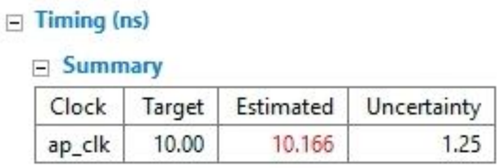
	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
YOLO2_FPGA	0.88	180	150	32188	57235		undef	none
intra_pingpong_wrapp	0.12	36	139	24092	43613	1~334379522	1 ~ 334379522	none
compute4	-	0	130	11546	18987	3~16392	3 ~ 16392	none
copy_local_beta	-	0	0	1133	1274	2~34	2 ~ 34	none
copy_input_weight	0.12	4	3	2823	7879	1~1044933	1 ~ 1044933	none
input_load	0.12	0	3	1772	3782	7~1044931	7 ~ 1044931	none
mmcpy_inp	-	0	0	641	1272	1~45	1 ~ 45	none
copy_input2	-	0	0	394	1203	1~16324	1 ~ 16324	none
weight_load_rec	-	4	0	948	3868	1~5369	1 ~ 5369	none
pool_yolo26	-	0	2	537	830	1~1959	1 ~ 1959	none
reorg_yolo25	-	0	4	403	711	1~1926	1 ~ 1926	none
write_back_output_re	0.88	4	8	1496	2752	1~367667	1 ~ 367667	none
outputpixel2buf	0.88	0	5	784	1222	1~849	1 ~ 849	none
mmcpy_outputpixe	-	0	0	178	467	1~264	1 ~ 264	none
mmcpy_output	-	0	0	87	198	1~263	1 ~ 263	none
mmcpy_output	-	0	0	87	198	1~263	1 ~ 263	none
copy_beta	-	1	0	263	341	11~1073742347	11 ~ 1073742347	none



	Negative Slack	BRAM	DSP	FF	LUT	Latency	Interval	Pipeline type
YOLO2_FPGA	0.88	184	150	32200	57352		undef	none
intra_pingpong_wrapp	0.12	40	139	24104	43730	1~70208002	1 ~ 70208002	none
compute4	-	0	130	11546	18987	3~16392	3 ~ 16392	none
copy_input_weight	0.12	8	3	2835	7996	1~219397	1 ~ 219397	none
input_load	0.12	4	3	1784	3899	6~219395	6 ~ 219395	none
mmcpy_inp	-	0	0	641	1272	1~45	1 ~ 45	none
copy_input2	-	4	0	408	1328	1~3426	1 ~ 3426	none
weight_load_rec	-	4	0	948	3868	1~5369	1 ~ 5369	none
load_weight	-	0	0	687	843	1~67	1 ~ 67	none
weight_mm	-	0	0	190	288	1~266	1 ~ 266	none
pool_yolo26	-	0	2	537	830	1~1959	1 ~ 1959	none
reorg_yolo25	-	0	4	403	711	1~1926	1 ~ 1926	none
write_back_output_re	0.88	4	8	1496	2752	1~367667	1 ~ 367667	none
outputpixel2buf	0.88	0	5	784	1222	1~849	1 ~ 849	none
mmcpy_outputpixe	-	0	0	178	467	1~264	1 ~ 264	none
mmcpy_output	-	0	0	87	198	1~263	1 ~ 263	none
mmcpy_output	-	0	0	87	198	1~263	1 ~ 263	none
copy_beta	-	1	0	263	341	11~1073742347	11 ~ 1073742347	none

Optimization Summary

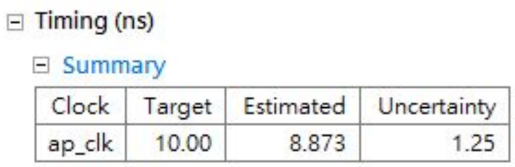
- Resolve timing violation by coding style and pragma.
- Calculate the weighted latency for different layer then decrease the critical latency.
- Raise the operating clock for a better FPS.



Timing (ns)

Summary

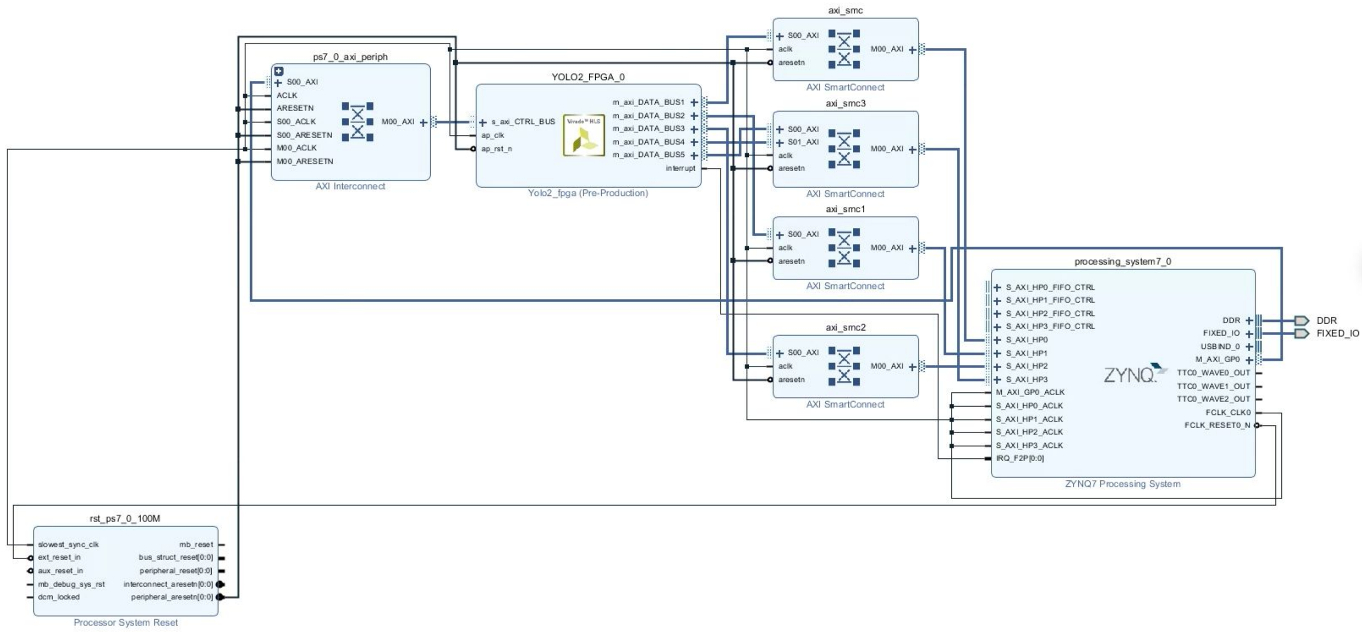
Clock	Target	Estimated	Uncertainty
ap_clk	10.00	10.166	1.25



Timing (ns)

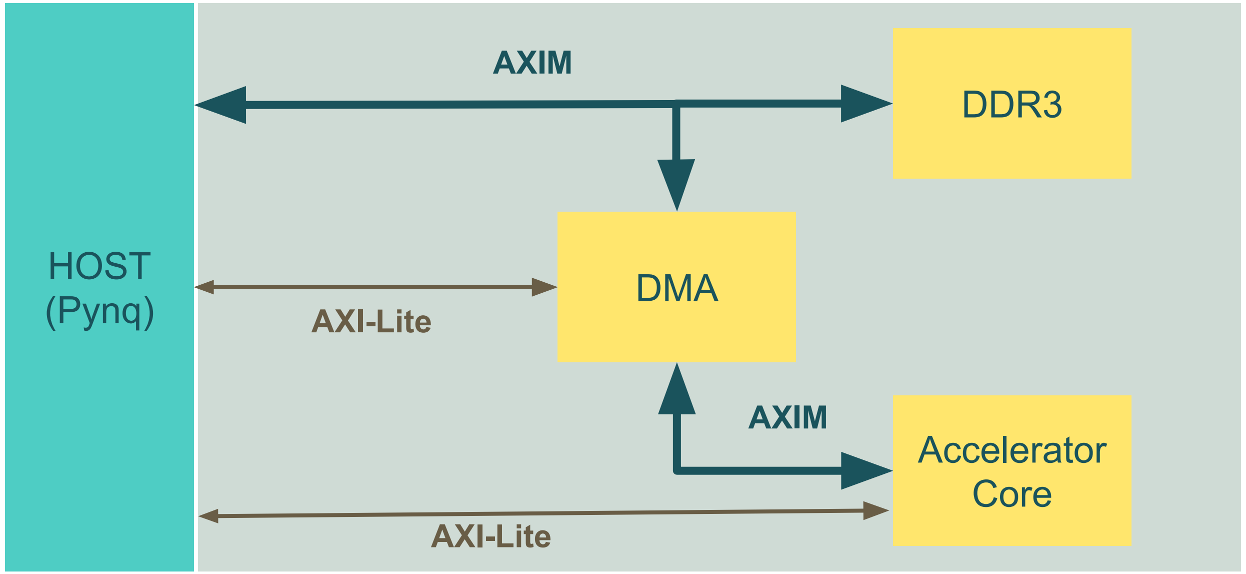
Summary

Clock	Target	Estimated	Uncertainty
ap_clk	10.00	8.873	1.25



Outline

System Block Design

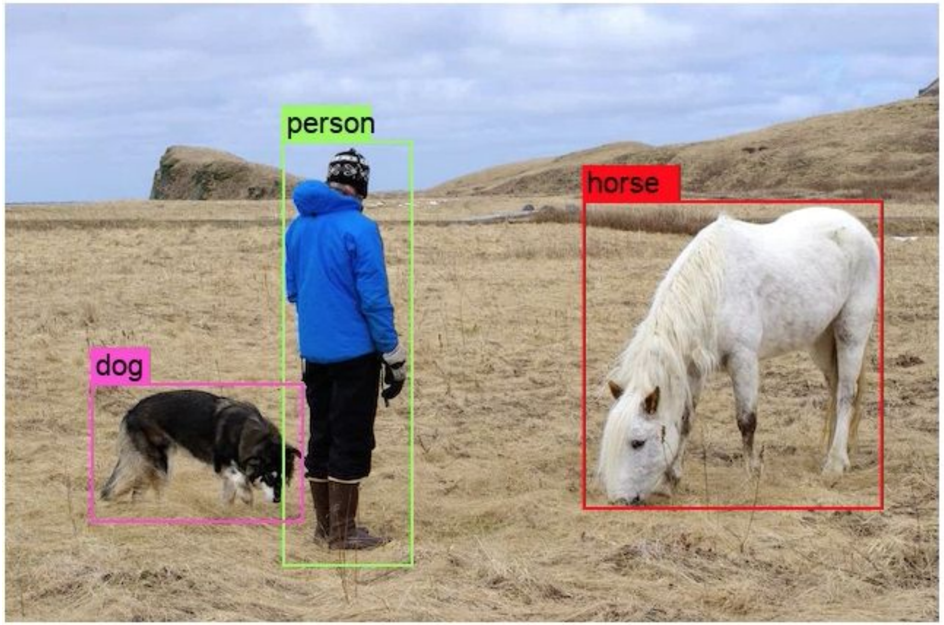


FPGA Verification

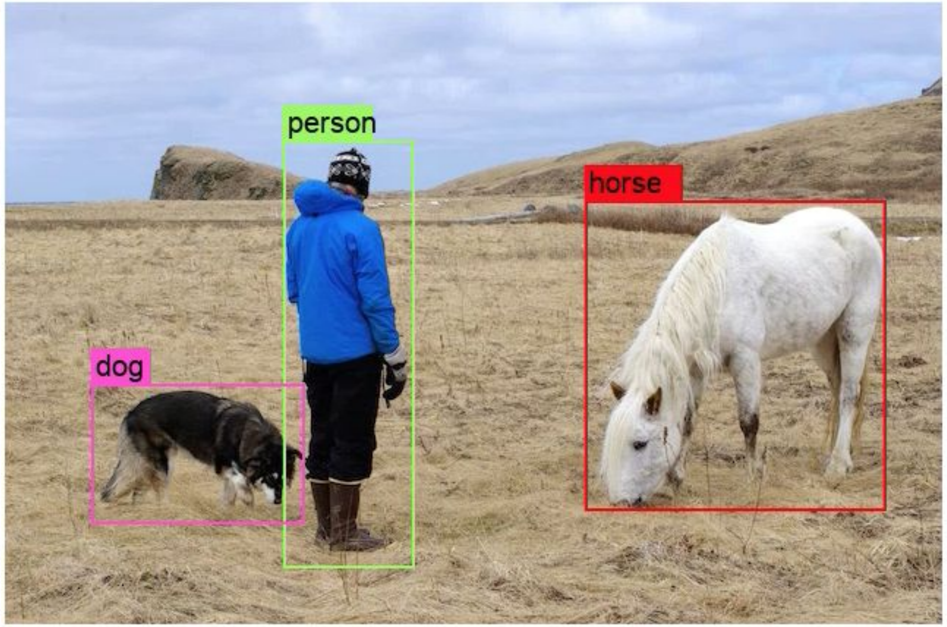
- Host: Pynq
- Read variables and image.
- Resize image to 416x416 to fit Yolov2.
- Prepare input image/weight and call accelerate core for each layers.
- After receiving data from HLS core, postprocess and draw the detection.

Results

```
image_preprocess          : 0.21821856498718262
load image to memory time: 0.02039051055908203
fpga_process_time         : 2.6969799995422363
region_layer_process_time: 0.5516119003295898
post_process_time         : 0.5308983325958252
```

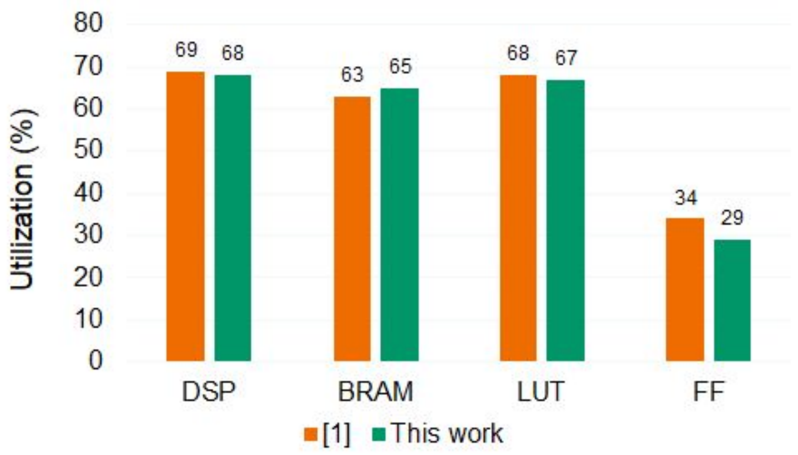


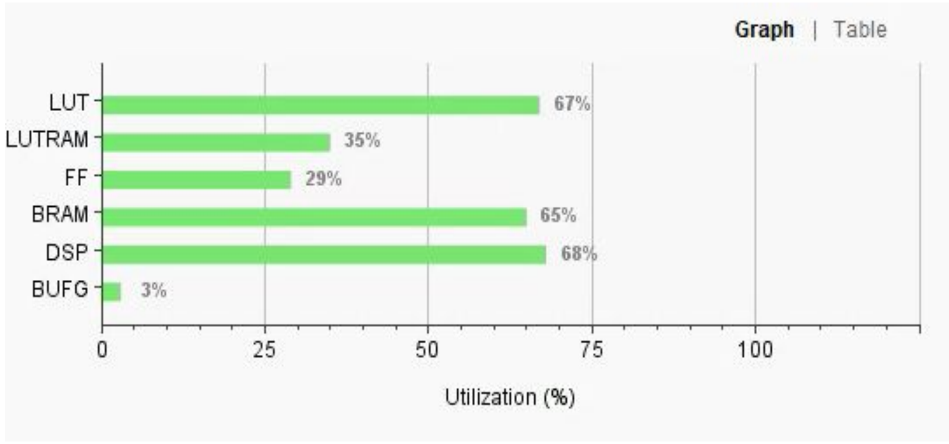
```
image_preprocess          : 0.21631407737731934
load image to memory time: 0.0202791690826416
fpga_process_time         : 1.293474674224853
region_layer_process_time: 0.550321102142334
post_process_time         : 0.5327498912811279
```



Performance Comparison

Resource						
	DSP	**BRAM**	**LUT**	**FF**	**Frequency**	**Device**
Baseline	69%	63%	68%	34%	100MHz	Zedboard
This work	68%	65%	67%	29%	125MHz	Zedboard





Performance Comparison (Cont'd)

Performance		
	Baseline	**This work**
CNN models	YOLOv2	YOLOv2
Clock (MHz)	100	100 / 125
Precision	Fixed-16	Fixed-16
Power (W)	2.25	2.31
Operations (GOP)	29.47	29.47
FPGA processing time (S)	2.69	1.58 / 1.29
Performance (GOP/sec)	10.96	18.65 / 22.84
Power efficiency (GOP/sec/W)	4.87	8.07 / 9.89

Thank you

Reference

[1] *https://github.com/dhm2013724/yolov2_xilinx_fpga*

[2] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger. In Computer Vision and Pattern Recognition (CVPR), *2017 IEEE Conference on, pages 6517–6525. IEEE, 2017*

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385,2015.*

[4] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *arXiv preprint arXiv:1506.02640, 2015.*

[5] Seong, & Song, Shengli & Yoon, Jin-Ha & Kim, Jungsun & Choi, Chil-Sung. (2019). Determination of Vehicle Trajectory through Optimization of Vehicle Bounding Boxes Using a Convolutional Neural Network*. Sensors. 19. 4263. 10.3390/s19194263.*

[6] Z. Zhao, K. M. Barijough and A. Gerstlauer, "DeepThings: Distributed Adaptive Deep Learning Inference on Resource-Constrained IoT Edge Clusters," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 11, pp. 2348-2359, Nov. 2018, doi: 10.1109/TCAD.2018.2858384.